SELECT AE COURSE – OPTIONS ARE

SELECT YOUR AE COURSE

186 responses

BCS358C-
Project Management with Git

Version Control with Git, 3rd Edition, by Prem Kumar Ponuthorai, Jon Loeliger Released October 2022, Publisher(s): O'Reilly Media, Inc.

- *Pro Git book, written by Scott Chacon and Ben Straub and published by Apress, https://gitscm.com/book/en/v2*
- *https://infyspringboard.onwingspan.com/web/en/app/toc/lex_auth_01309444334736998427 82_shared/overview*
- *https://infyspringboard.onwingspan.com/web/en/app/toc/lex_auth_01330134712177459211926_shared/overview*

1154..0099..22002233

(As per VTU Syllabus)

Program Outcomes:

PROGRAM OUTCOMES

PO's	PO Description
PO1	**Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO2	**Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO3	**Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO4	**Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO5	**Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
PO6	**The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO7	**Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO8	**Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

Program Outcomes:

PO9	**Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO10	**Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO11	**Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO12	**Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes:

PROGRAM SPECIFIC OUTCOMES

PSO's	PSO Description
PSO1	An ability to design and analyze algorithms by applying theoretical concepts to build complex and computer- based systems in the domain of System Software, Computer Networks & Security, Web technologies, Data Science and Analytics.
PSO2	Be able to develop various software solutions by applying the techniques of Data Base Management, Complex Mathematical Models, Software Engineering practices and Machine Learning with Artificial Intelligence.

BCS358C-Project Management with Git

CREDITS – 1

CIE (Continuous Internal Evaluation) MARKS – 50

SEE (Semester End Examination) MARKS – 50

BCS358C-Project Management with Git

Course objectives(CO):-

- To familiar with basic command of Git
- To create and manage branches
- To understand how to collaborate and work with Remote Repositories
- To familiar with version controlling commands

BCS358C-Project Management with Git

Course Outcomes:

- Use the basics commands related to git repository
- Create and manage the branches
- Apply commands related to Collaboration and Remote Repositories
- Use the commands related to Git Tags, Releases and advanced git operations
- Analyse and change the git history

BCS358C-Project Management with Git

BCS358C-Project Management with Git

Sl.NO	Experiments
1	**Setting Up and Basic Commands** Initialize a new Git repository in a directory. Create a new file and add it to the staging area and commit the changes with an appropriate commit message.
2	**Creating and Managing Branches** Create a new branch named "feature-branch." Switch to the "master" branch. Merge the "feature-branch" into "master."
3	**Creating and Managing Branches** Write the commands to stash your changes, switch branches, and then apply the stashed changes.
4	**Collaboration and Remote Repositories** Clone a remote Git repository to your local machine.
5	**Collaboration and Remote Repositories** Fetch the latest changes from a remote repository and rebase your local branch onto the updated remote branch.
6	**Collaboration and Remote Repositories** Write the command to merge "feature-branch" into "master" while providing a custom commit message for the merge.
7	**Git Tags and Releases** Write the command to create a lightweight Git tag named "v1.0" for a commit in your local repository.
8	**Advanced Git Operations**

BCS358C-Project Management with Git

	Write the command to cherry-pick a range of commits from "source-branch" to the current branch.
9	**Analysing and Changing Git History** Given a commit ID, how would you use Git to view the details of that specific commit, including the author, date, and commit message?
10	**Analysing and Changing Git History** Write the command to list all commits made by the author "JohnDoe" between "2023-01-01" and "2023-12-31."
11	**Analysing and Changing Git History** Write the command to display the last five commits in the repository's history.
12	**Analysing and Changing Git History** Write the command to undo the changes introduced by the commit with the ID "abc123".

What is Git?

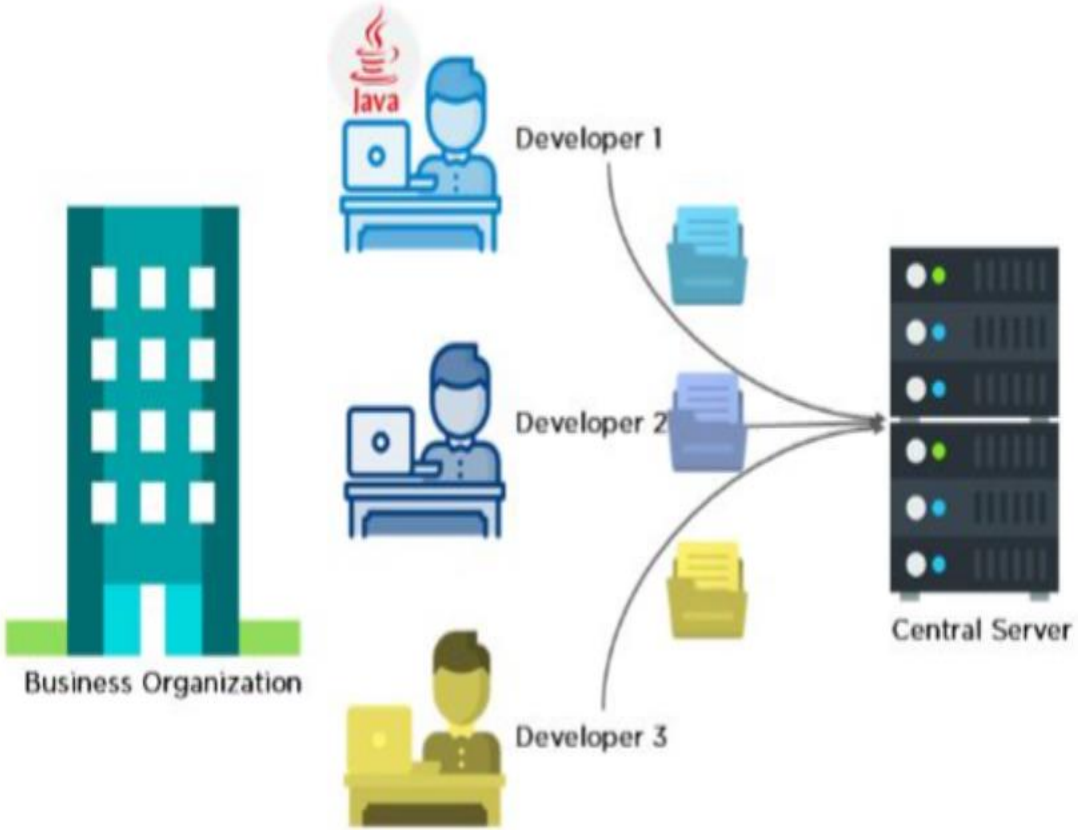
- Git is a version control system which lets you track changes you make to your files over time.
- With Git, you can revert to various states of your files.
- You can also make a copy of your file, make changes to that copy, and then merge these changes to the original copy.

Introduction to GIT

Git is a distributed version control system (VCS) that is widely used for tracking changes in source code during software development. It was created by Linus Torvalds in 2005 and has since become the de facto standard for version control in the software development industry.

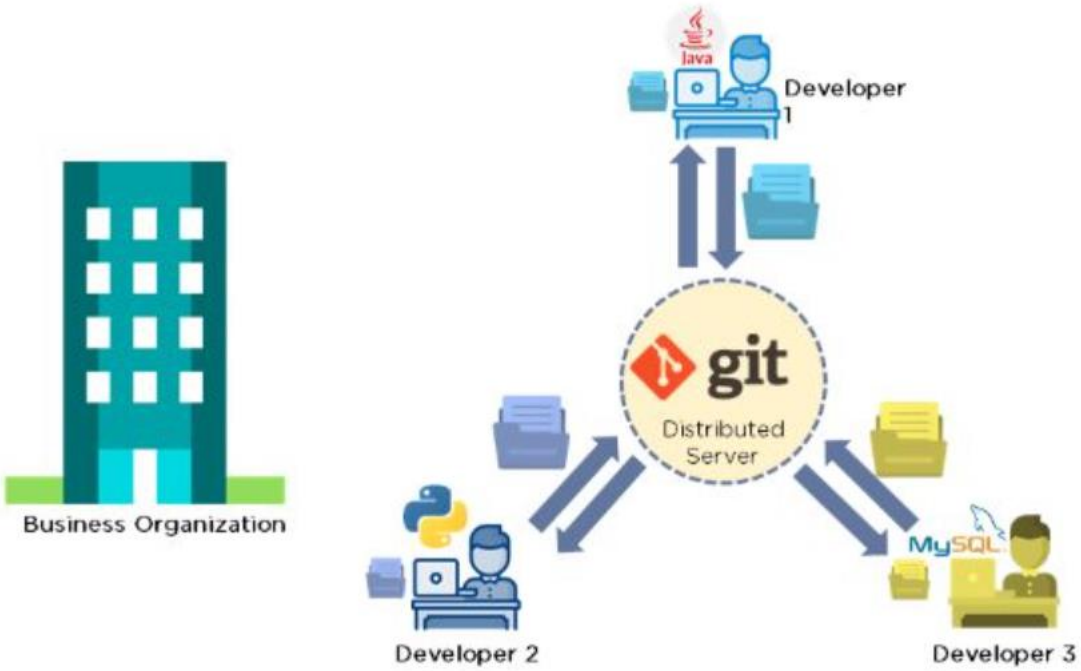
Before going through Git

- Developers used to submit their codes to the central server without having copies of their own
- Any changes made to the source code were unknown to the other developers
- There was no communication between any of the developers



After Pass through the Git

- Every developer has an entire copy of the code on their local systems
- Any changes made to the source code can be tracked by others
- There is regular communication between the developers



Introduction to GIT

- **Repository (Repo):** A Git repository is a directory or storage location where your project's files and version history are stored. There can be a local repository on your computer and remote repositories on servers.
- **Commits:** In Git, a commit is a snapshot of your project at a particular point in time. Each commit includes a unique identifier, a message describing the changes, and a reference to the previous commit.
- **Branches**: Branches in Git allow you to work on different features or parts of your project simultaneously without affecting the main development line (usually called the "master" branch). Branches make it easy to experiment, develop new features, and merge changes back into the main branch when they are ready.

Introduction to GIT

- **Pull Requests (PRs):** In Git-based collaboration workflows, such as GitHub or GitLab, pull requests are a way for developers to propose changes and have them reviewed by their peers. This is a common practice for open-source and team-based projects.
- **Merging:** Merging involves combining changes from one branch (or multiple branches) into another. When a branch's changes are ready to be incorporated into the main branch, you can merge them.
- **Remote Repositories:** Remote repositories are copies of your project stored on a different server. Developers can collaborate by pushing their changes to a remote repository and pulling changes from it. Common remote repository hosting services include GitHub, GitLab, and Bitbucket.

Introduction to GIT

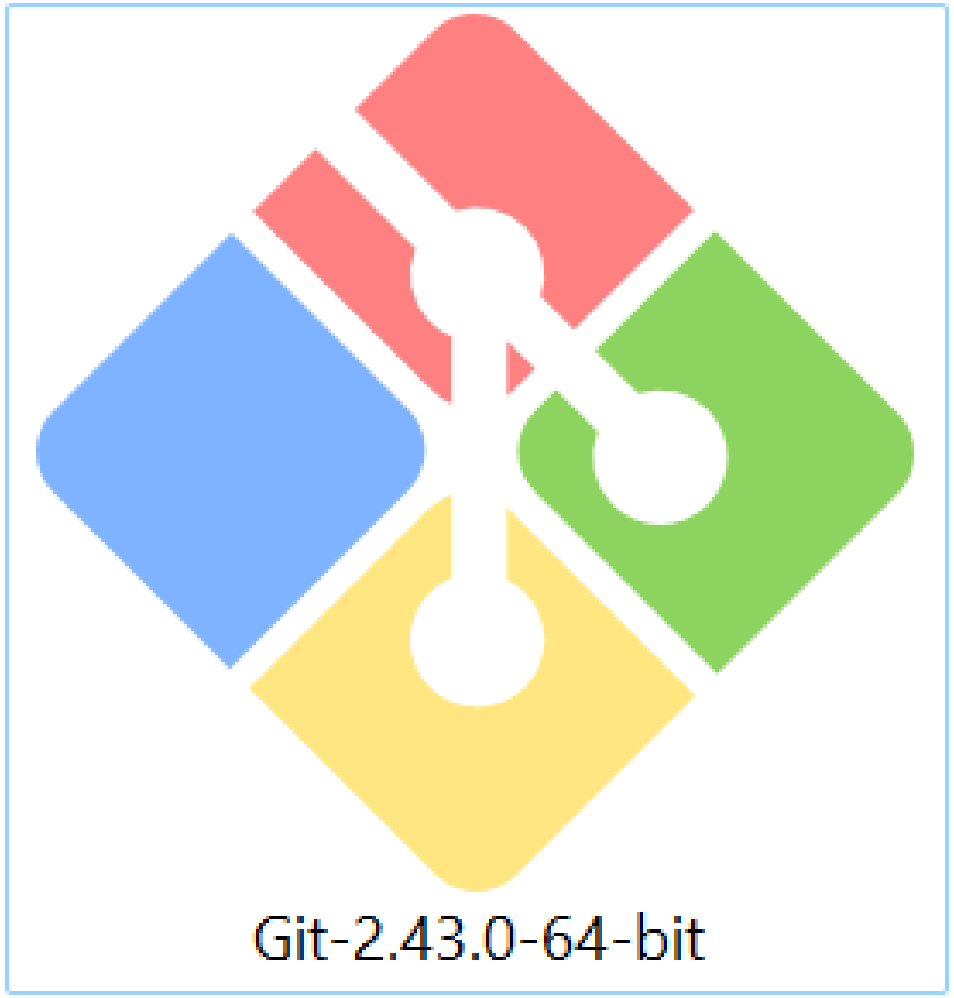
- **Cloning:** Cloning is the process of creating a copy of a remote repository on your local machine. This allows you to work on the project and make changes locally.
- **Forking:** Forking is a way to create your copy of a repository, typically on a hosting platform like GitHub. You can make changes to your fork without affecting the original project and later create pull requests to contribute your changes back to the original repository.

Features to GIT

- Version Control
- Collaboration
- Branching
- Distributed Development
- Backup and Recovery
- Code Review
- Open Source and Community Development

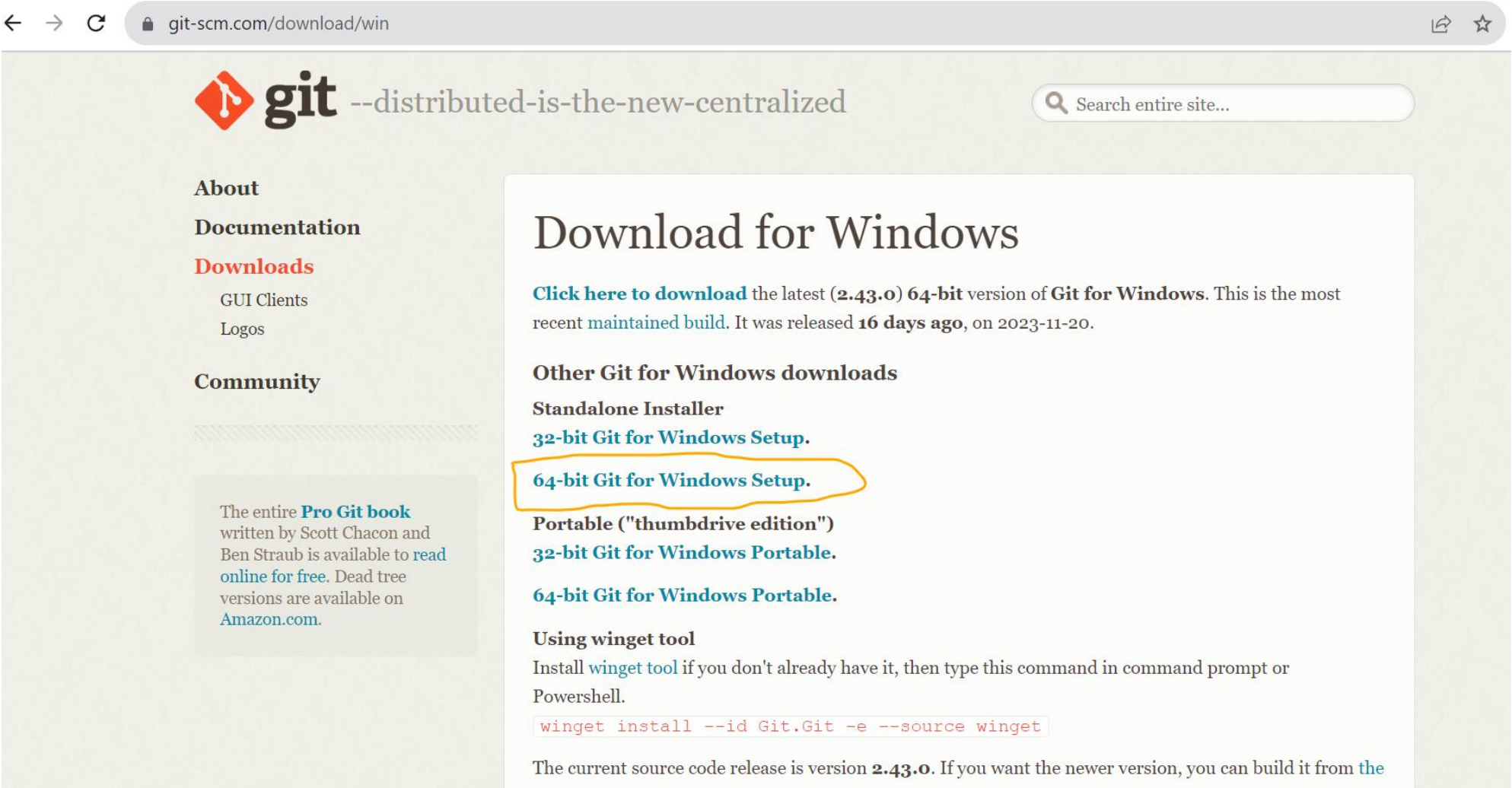
How to install Git

BCS358C-Project Management with Git

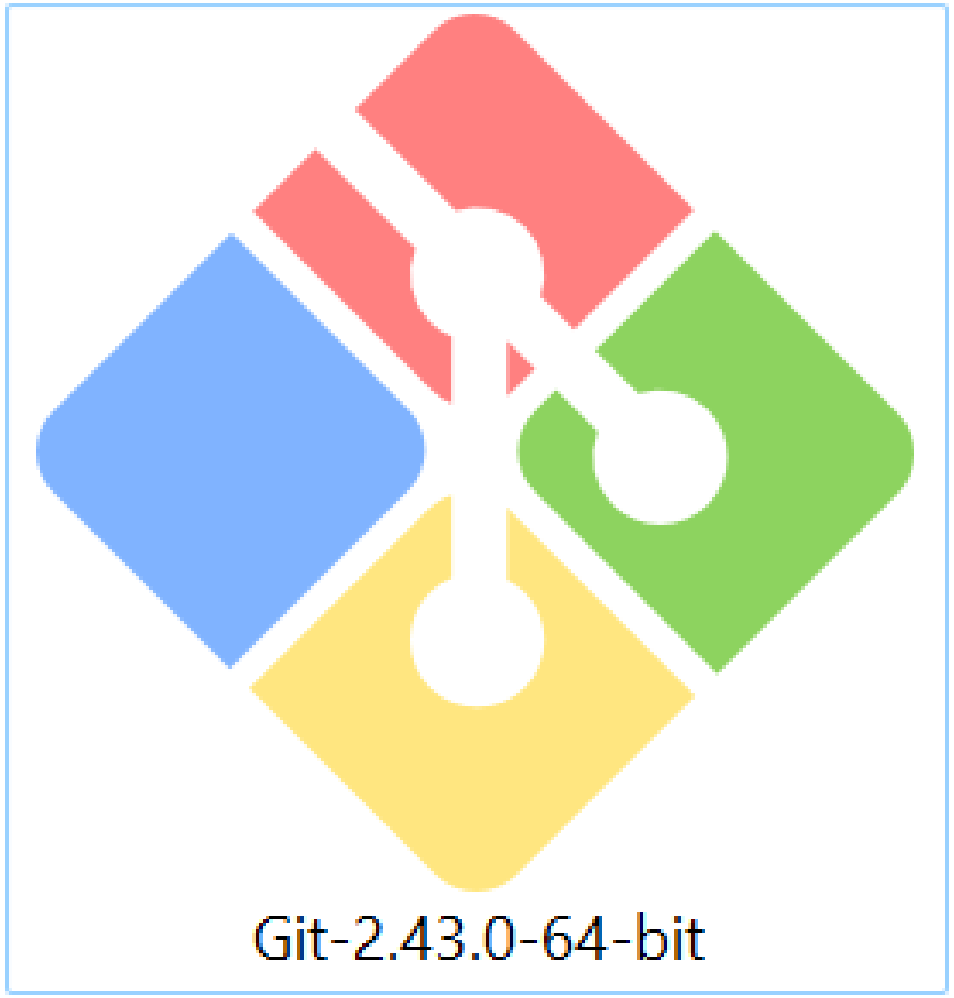


https://git-scm.com/download/win

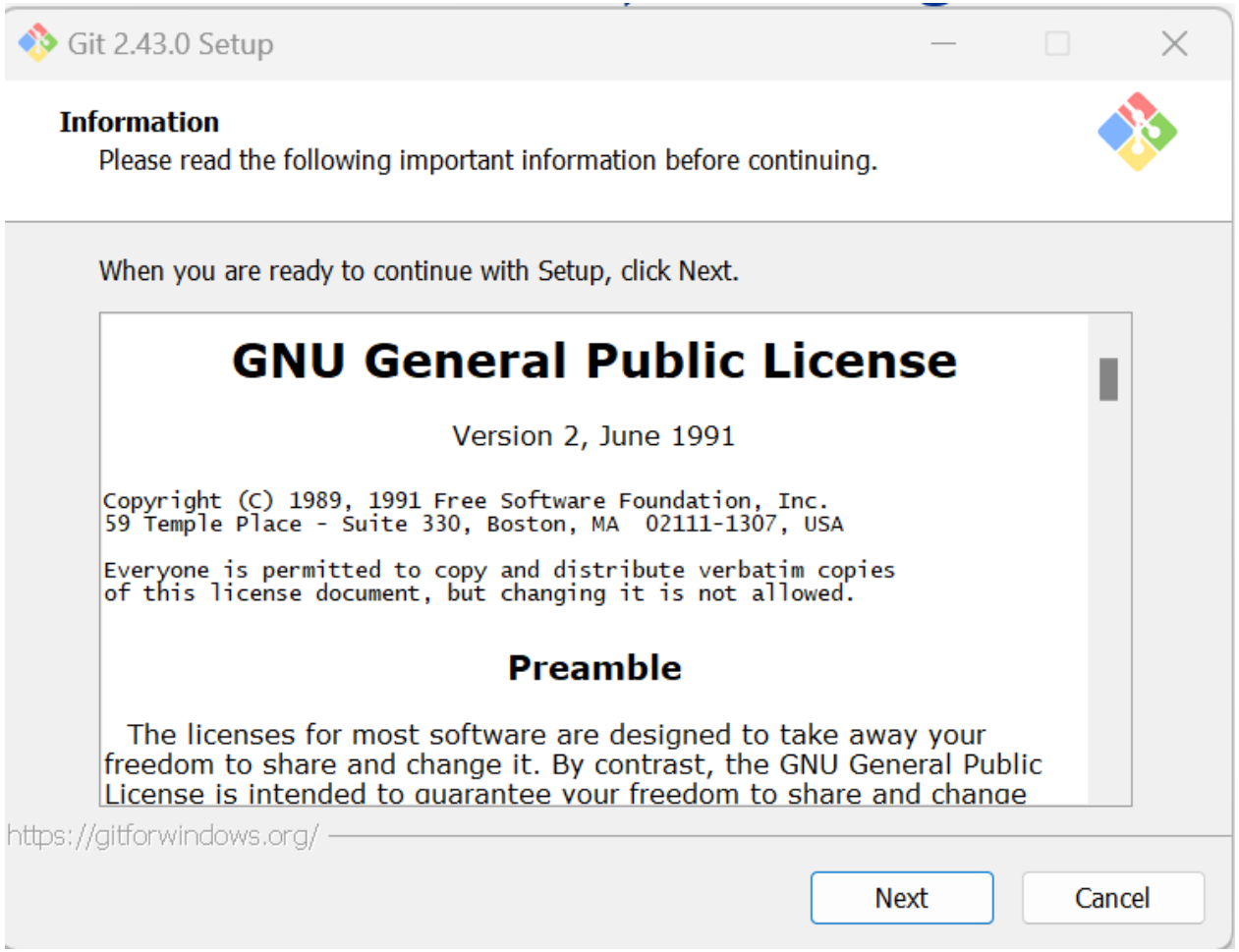
BCS358C-Project Management with Git



Click and download

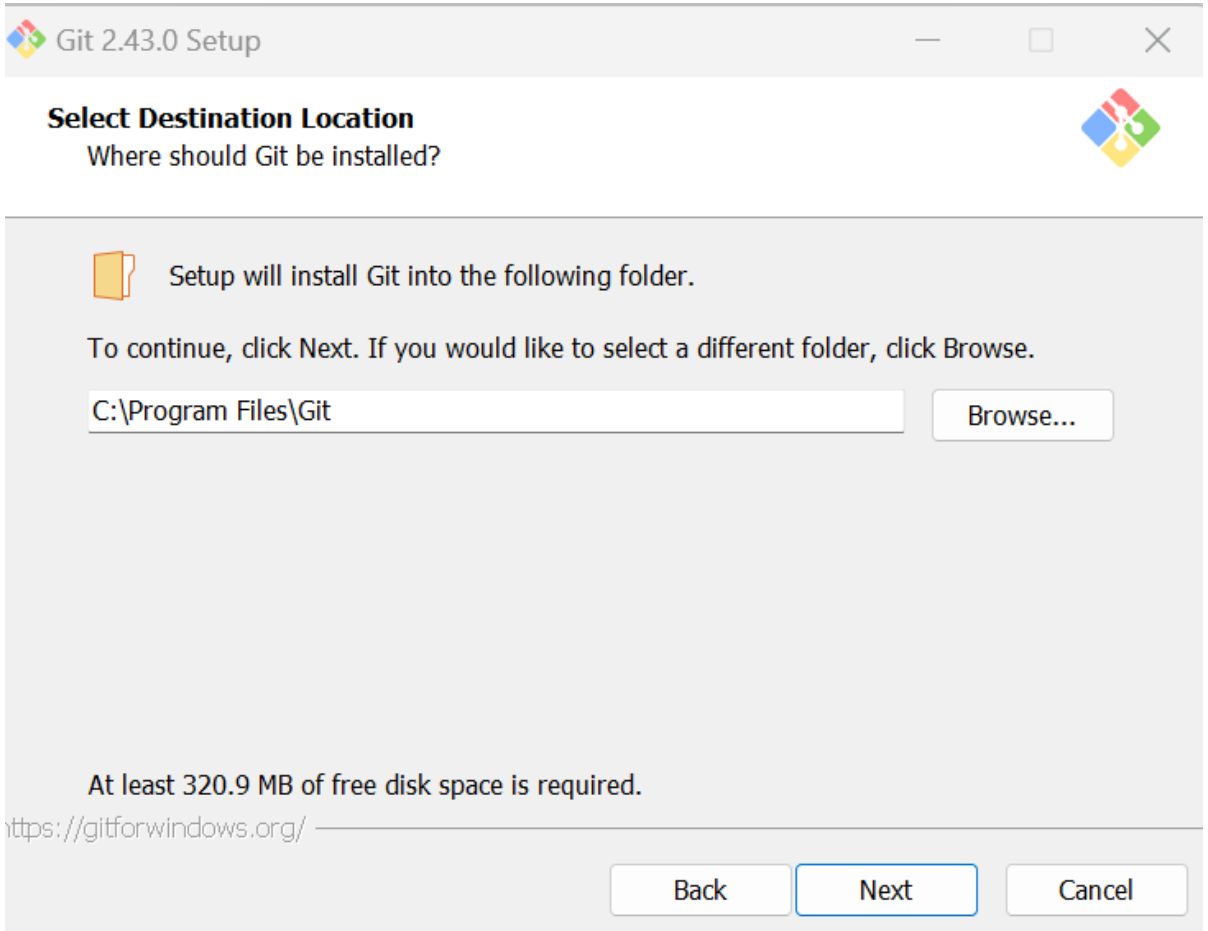


Check in your downloads
Right Click
And
Select
“Run As administrator”

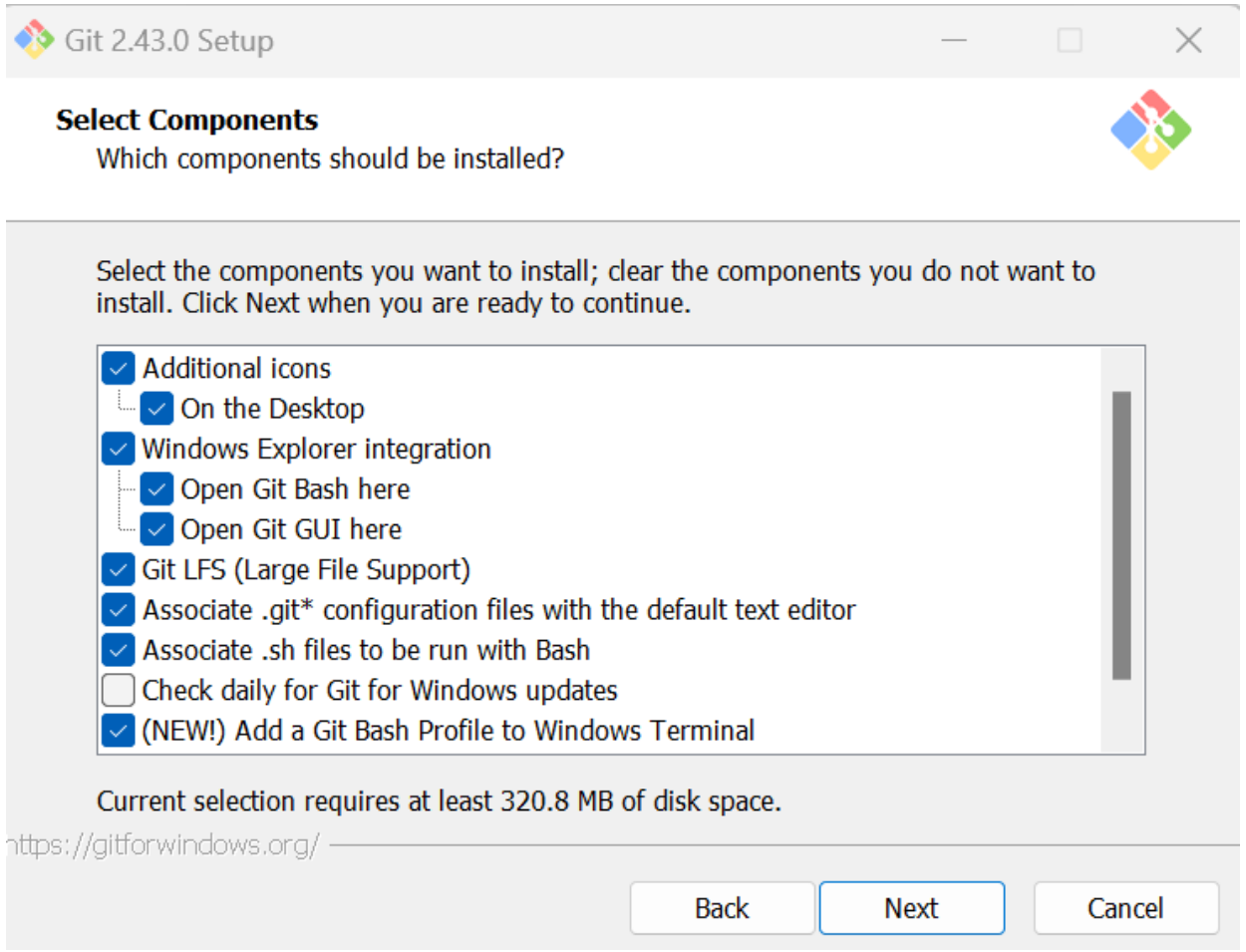

Git 2.43.0 Setup
Information
Please read the following important information before continuing.
When you are ready to continue with Setup, click Next.
GNU General Public License
Version 2, June 1991
Copyright (C) 1989, 1991 Free Software Foundation, Inc.
59 Temple Place - Suite 330, Boston, MA 02111-1307, USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.
Preamble
The licenses for most software are designed to take away your
freedom to share and change it. By contrast, the GNU General Public
License is intended to guarantee your freedom to share and change
https://gitforwindows.org/
Next
Cancel

Click Next

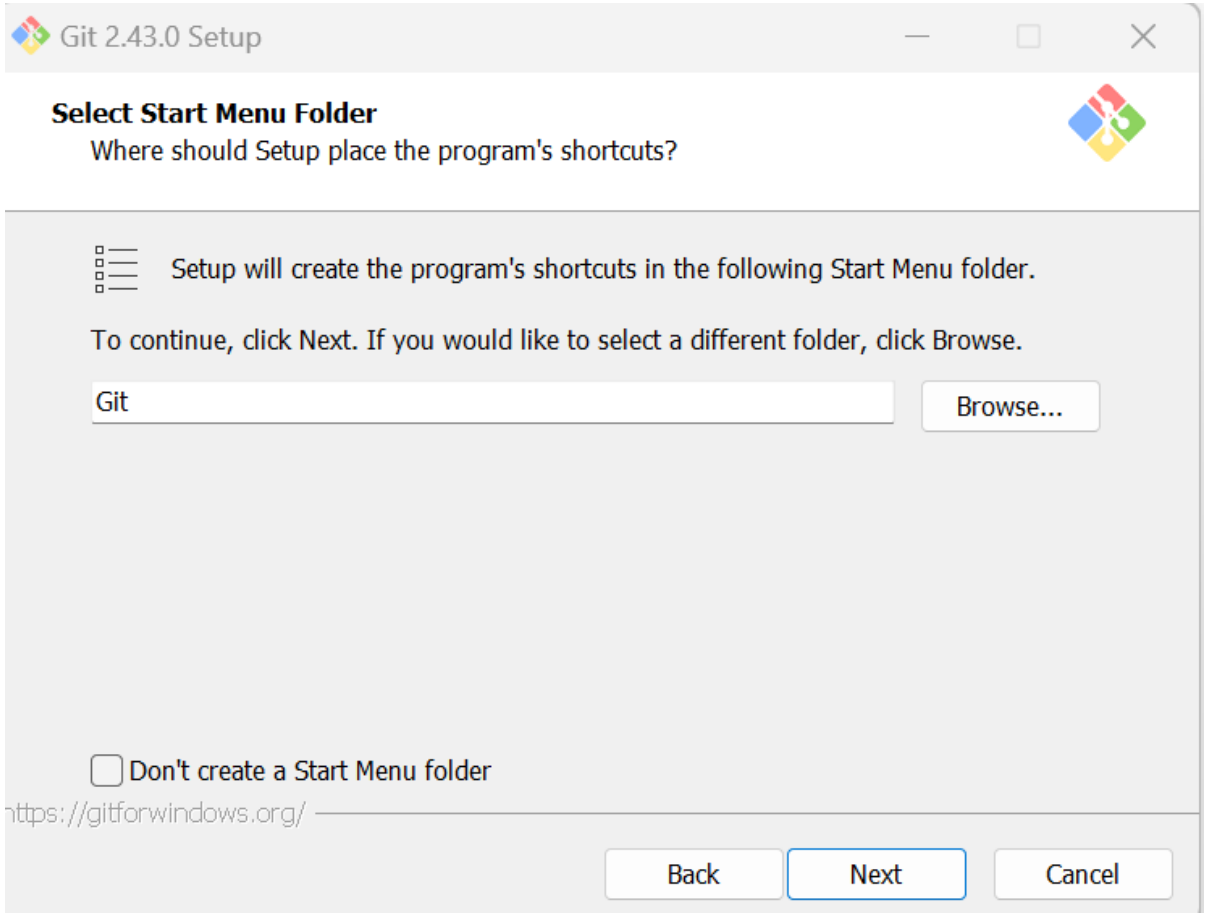
BCS358C-Project Management with Git



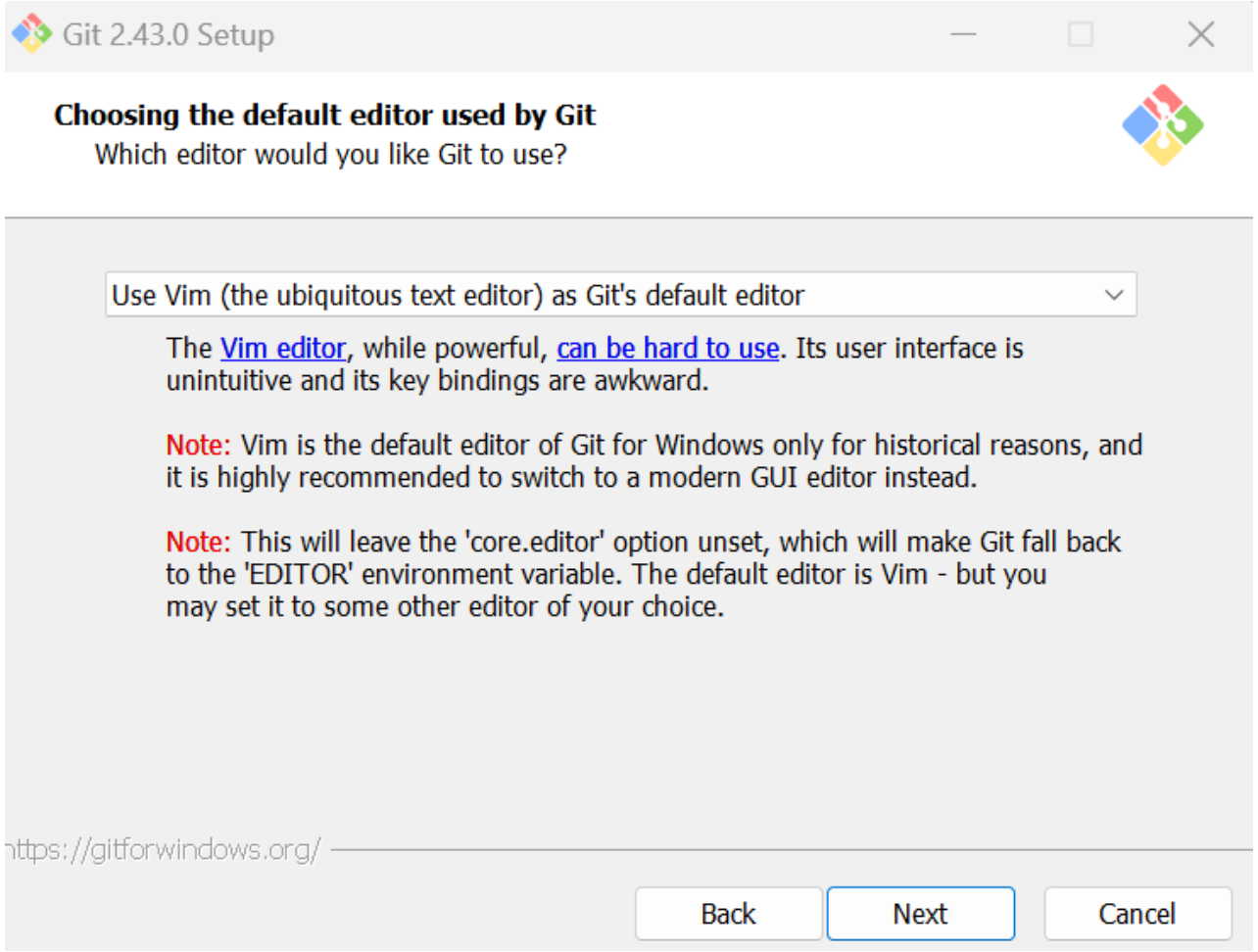
Click Next



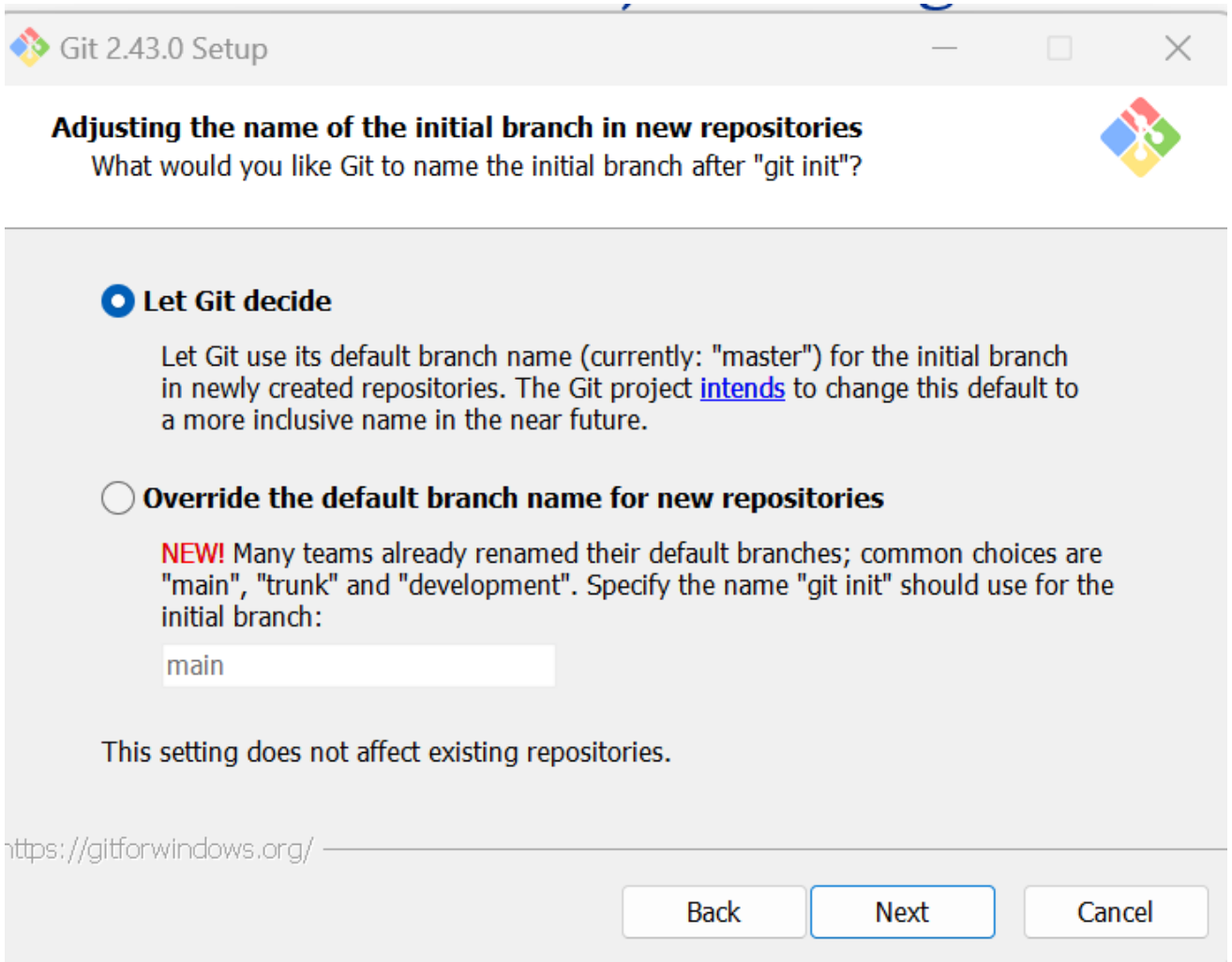
Select Additional icons & Click on Next



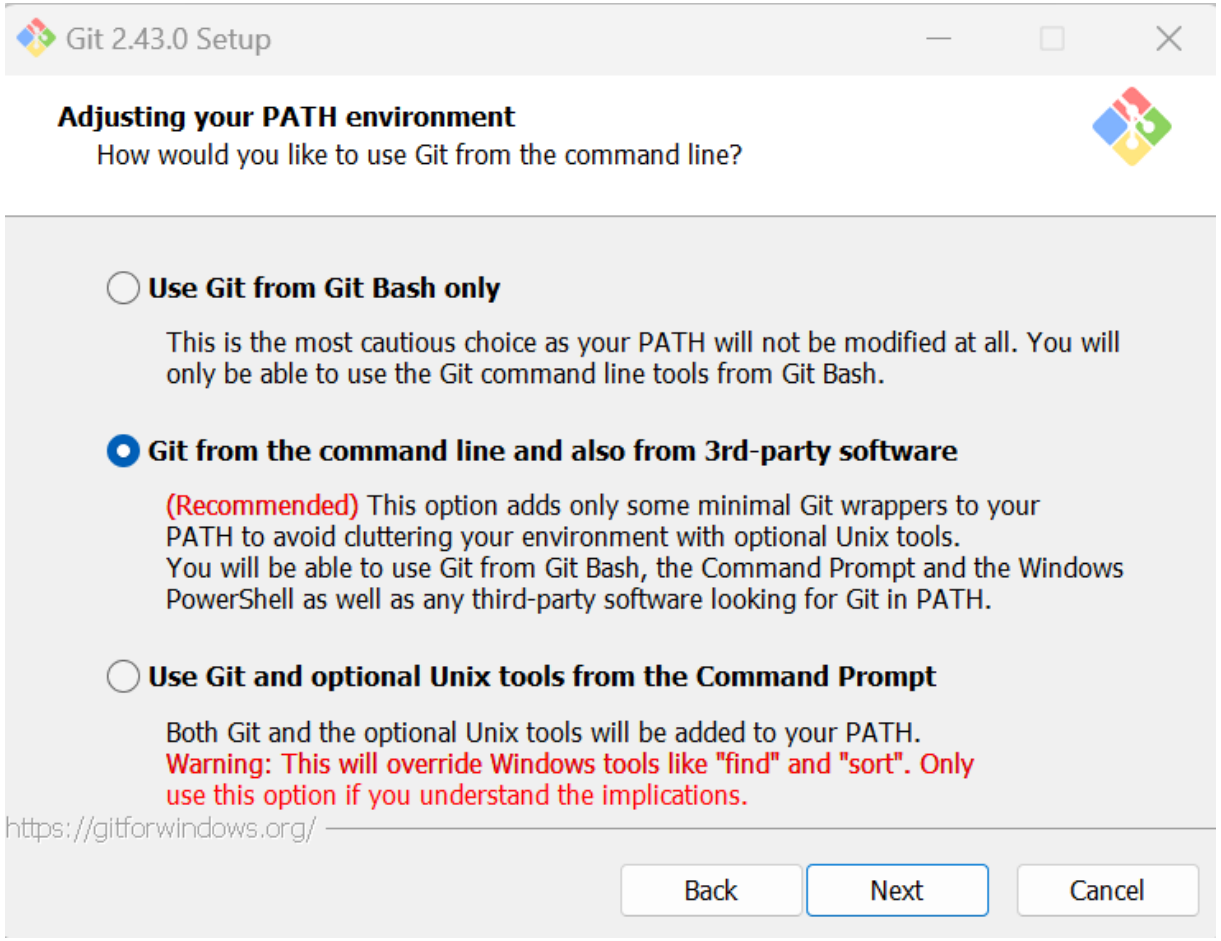
Click on Next



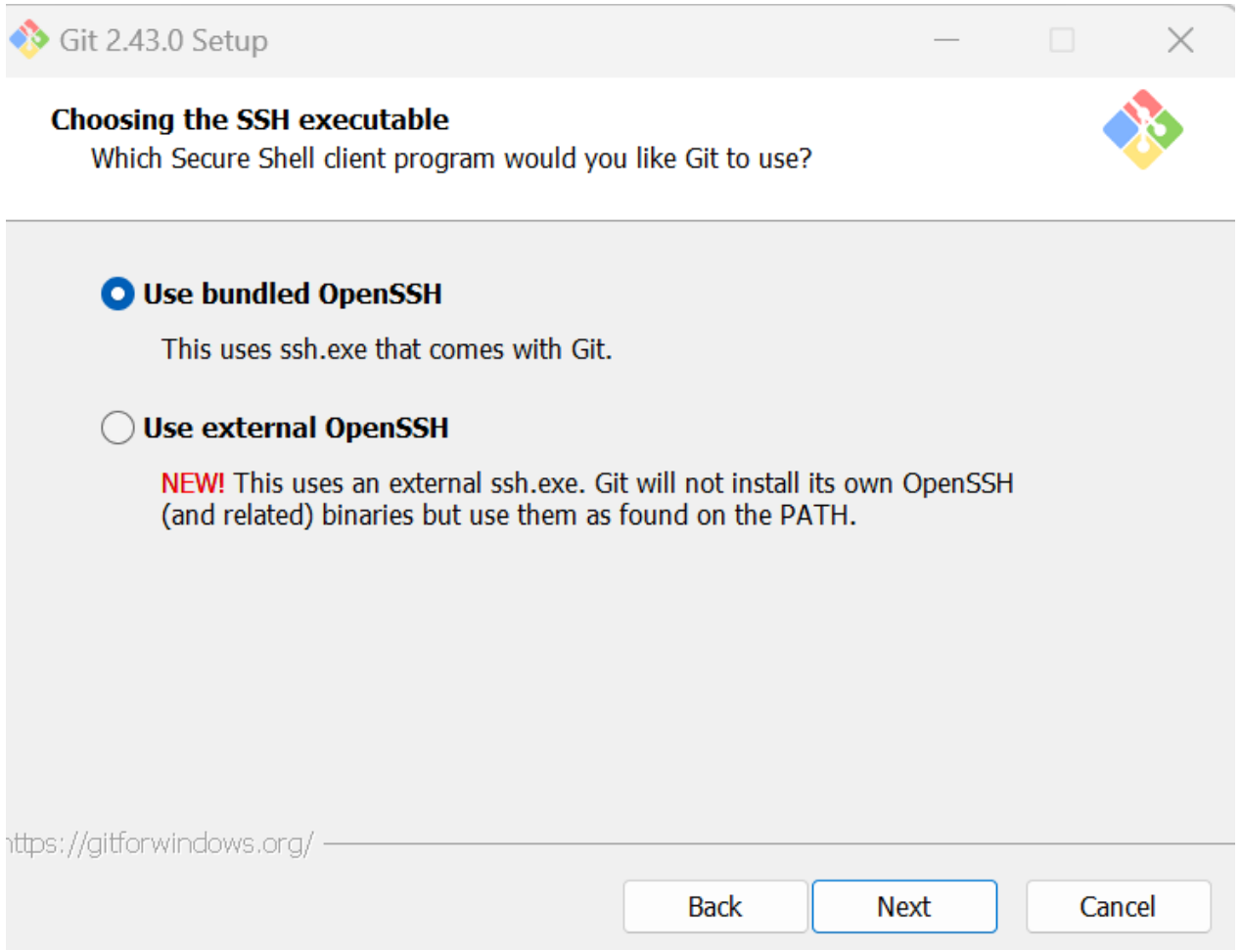
Click on Next



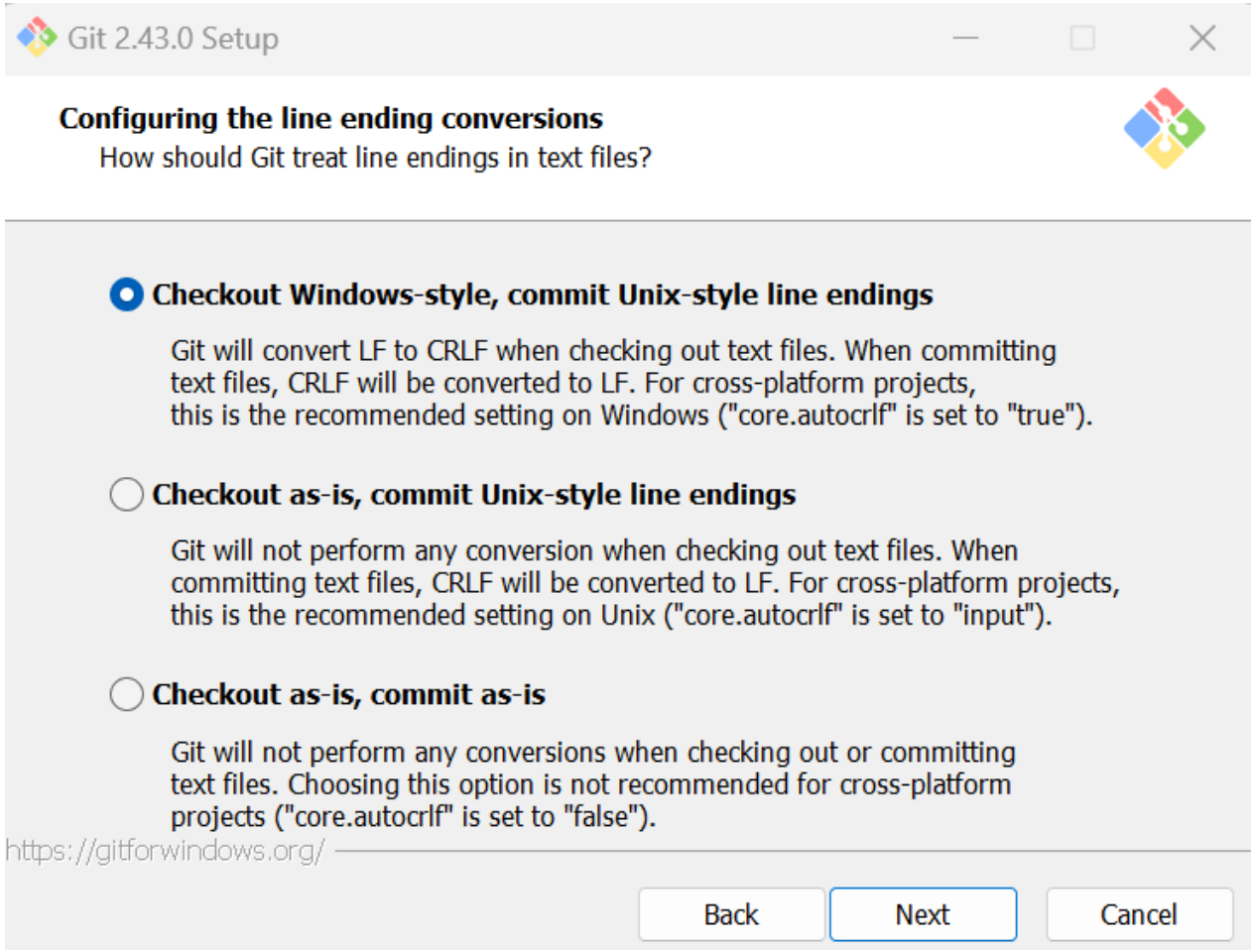
Click on Next



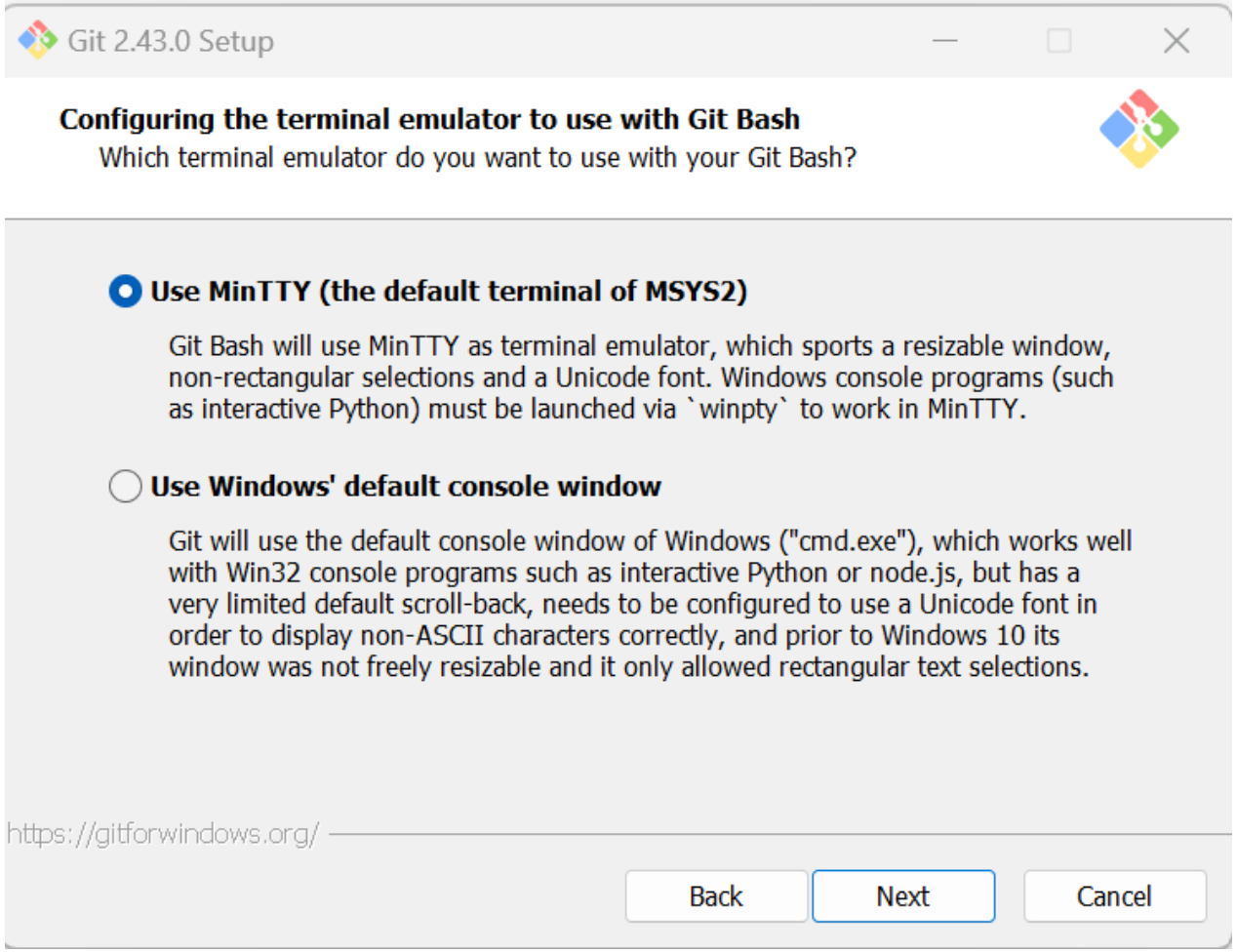
Click on Next



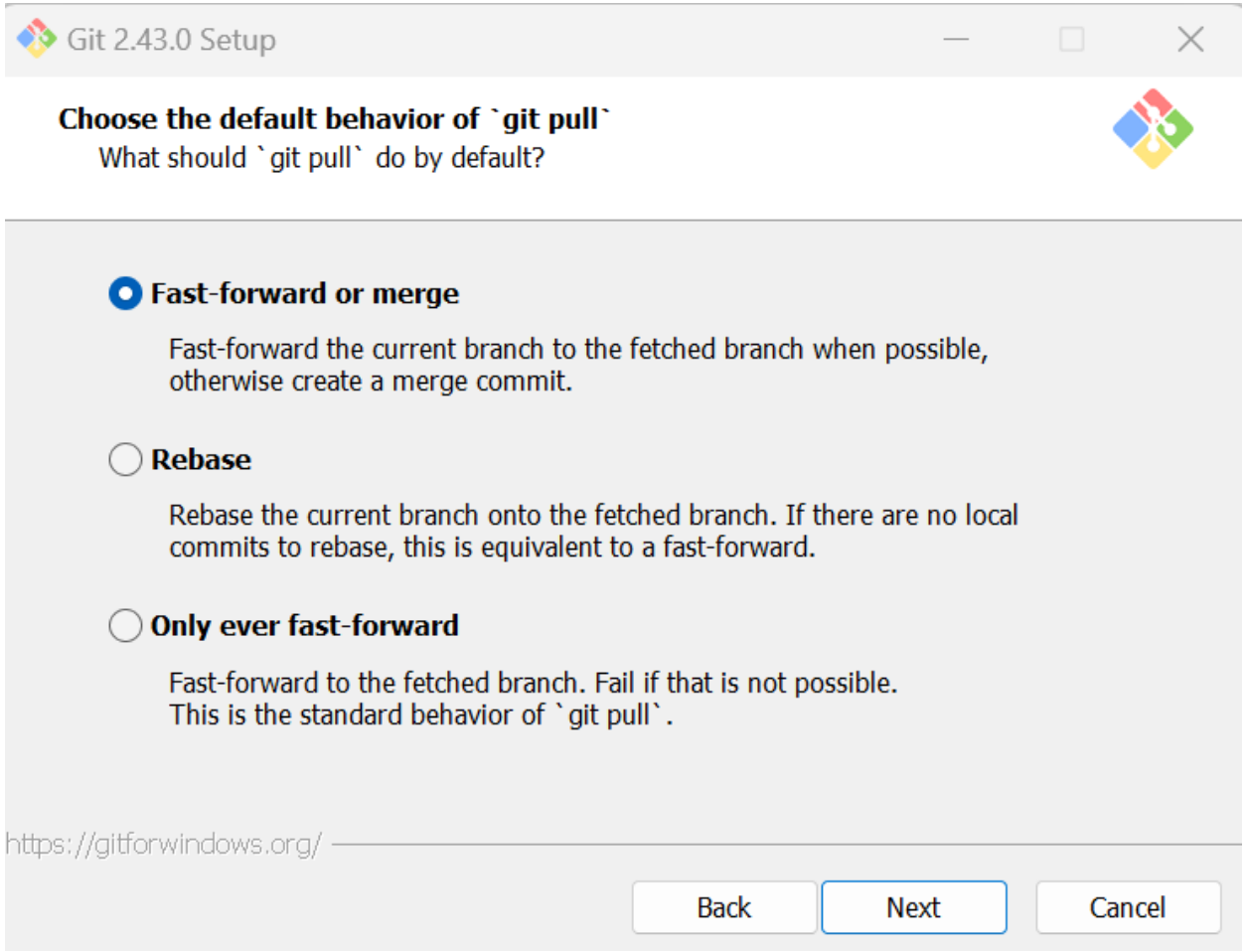
Click on Next



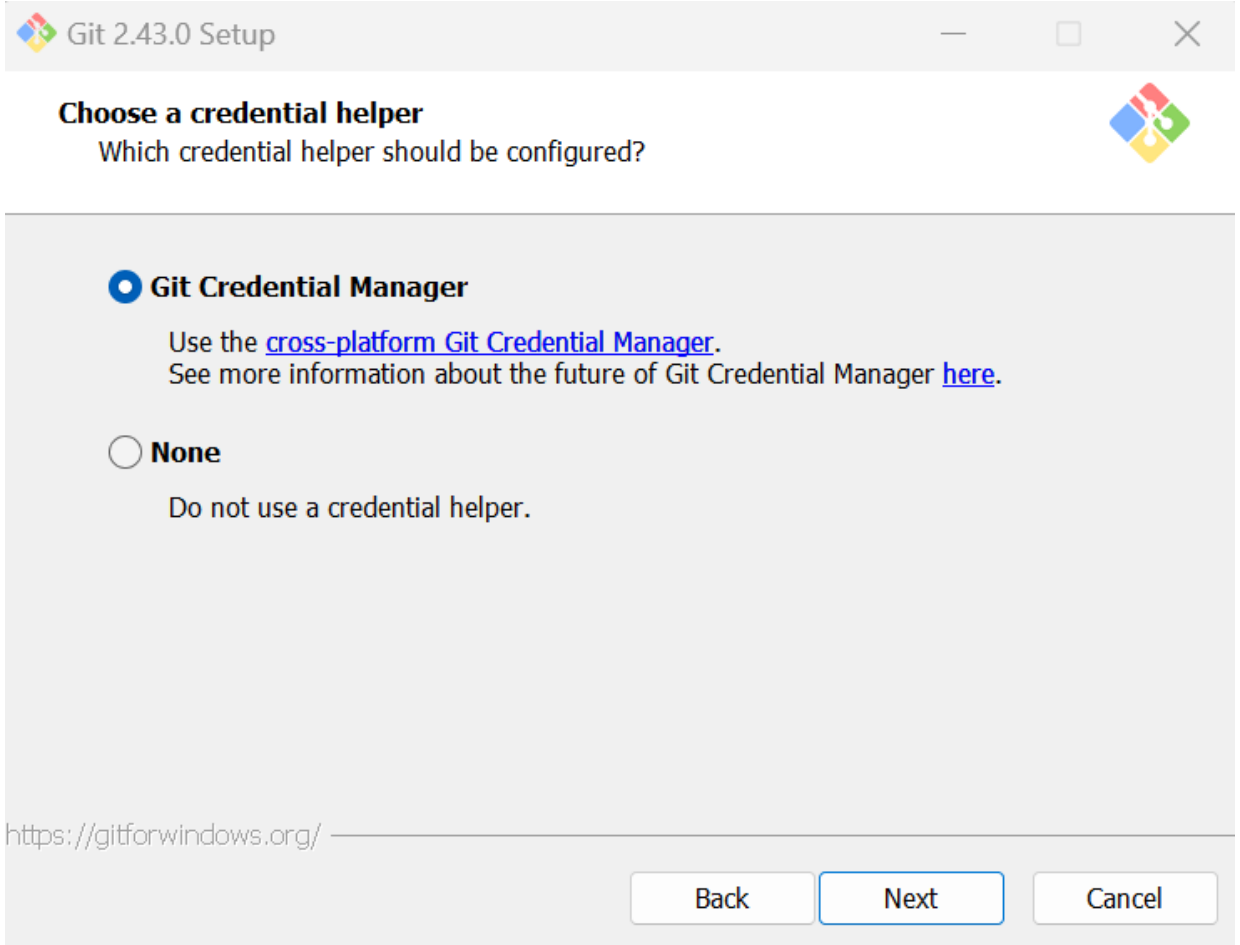
Click on Next



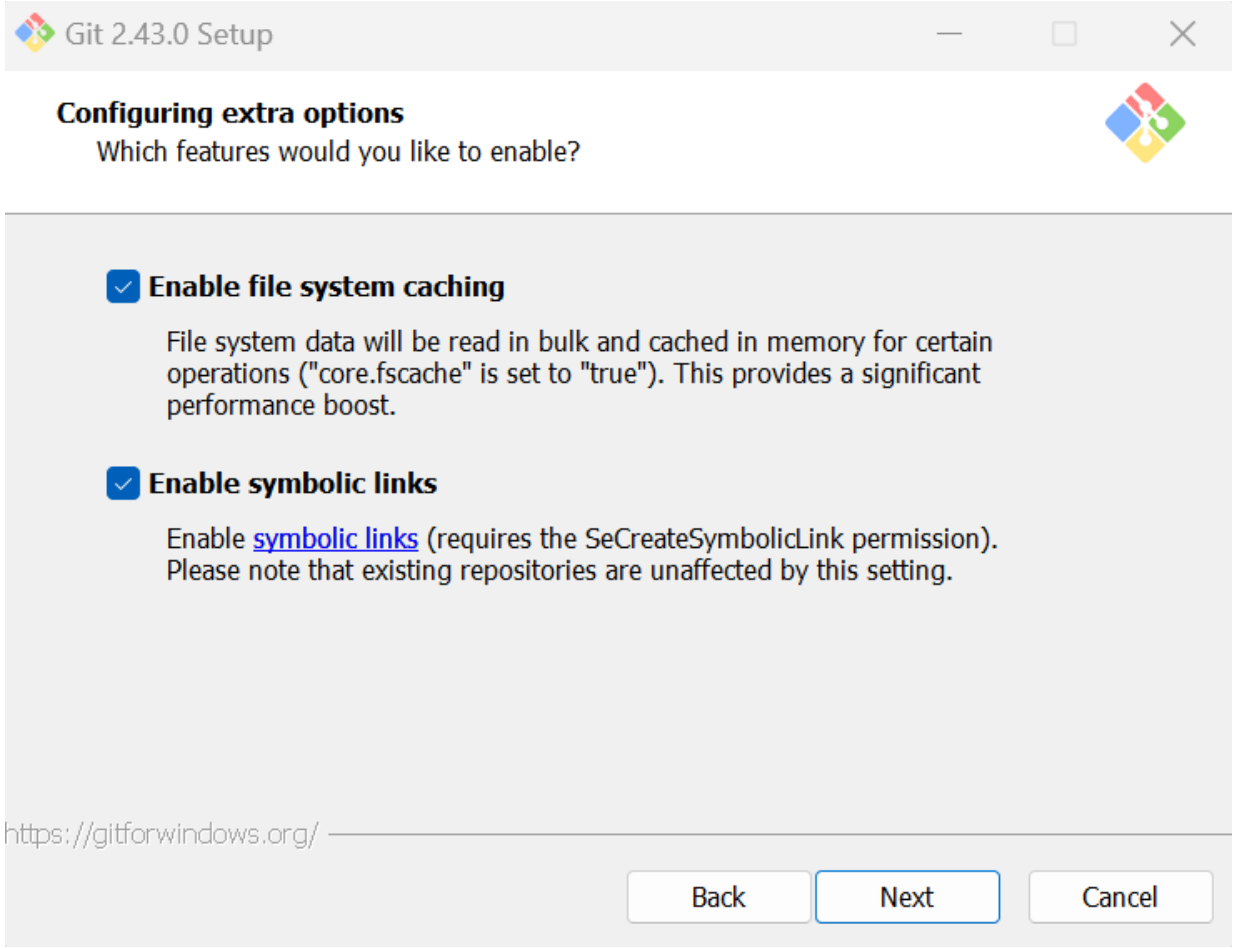
Click on Next



Click on Next

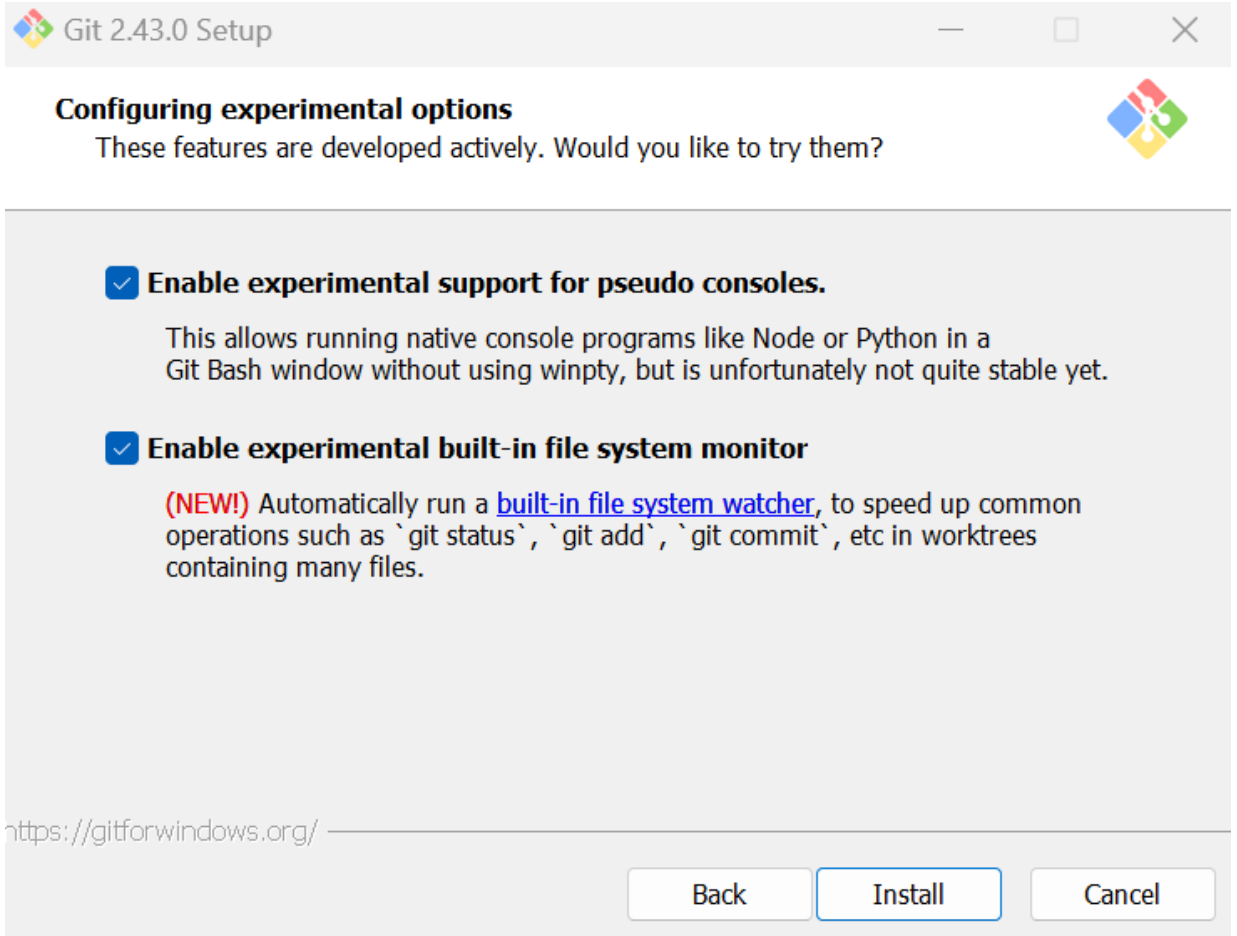


Click on Next

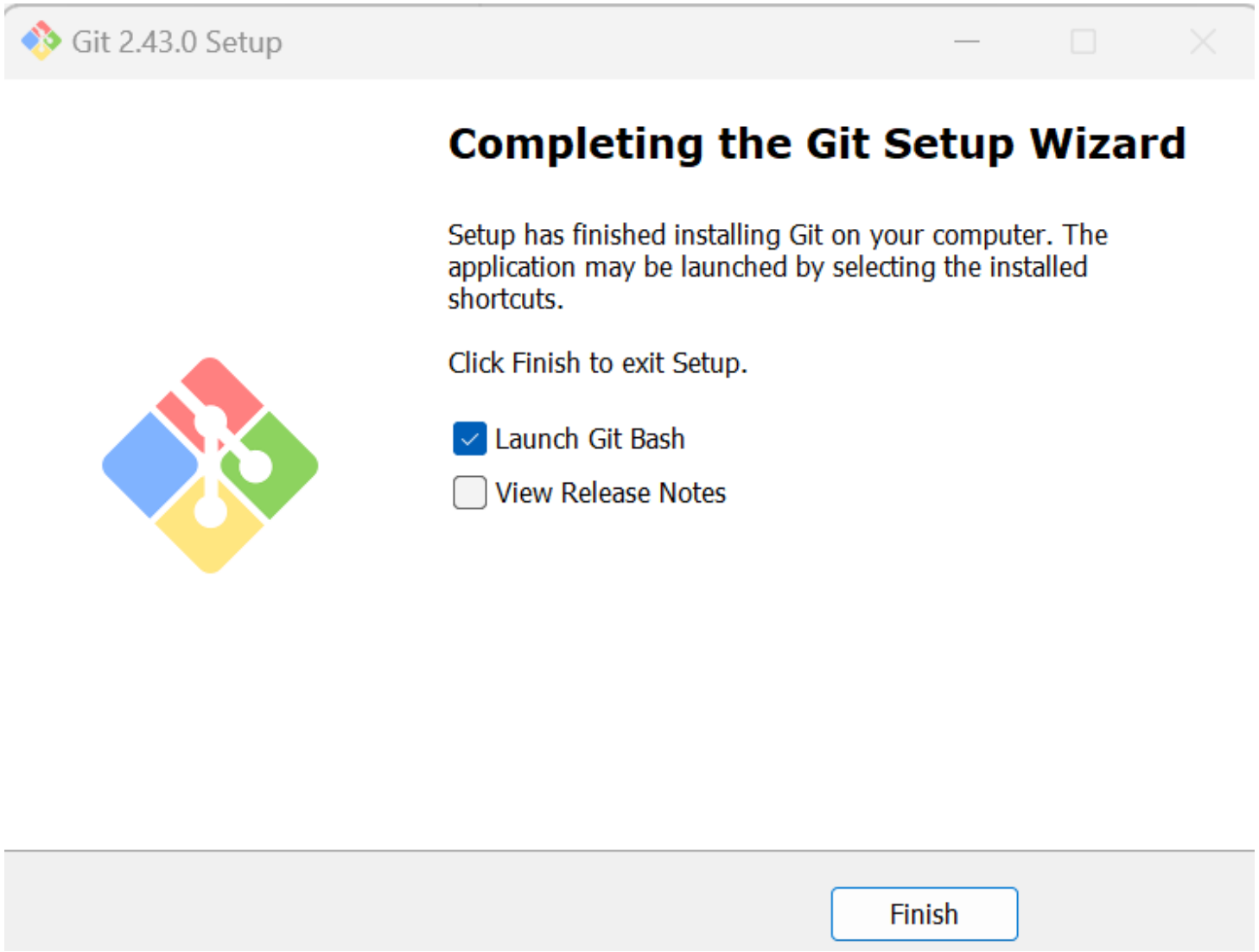


Click on Next

BCS358C-Project Management with Git

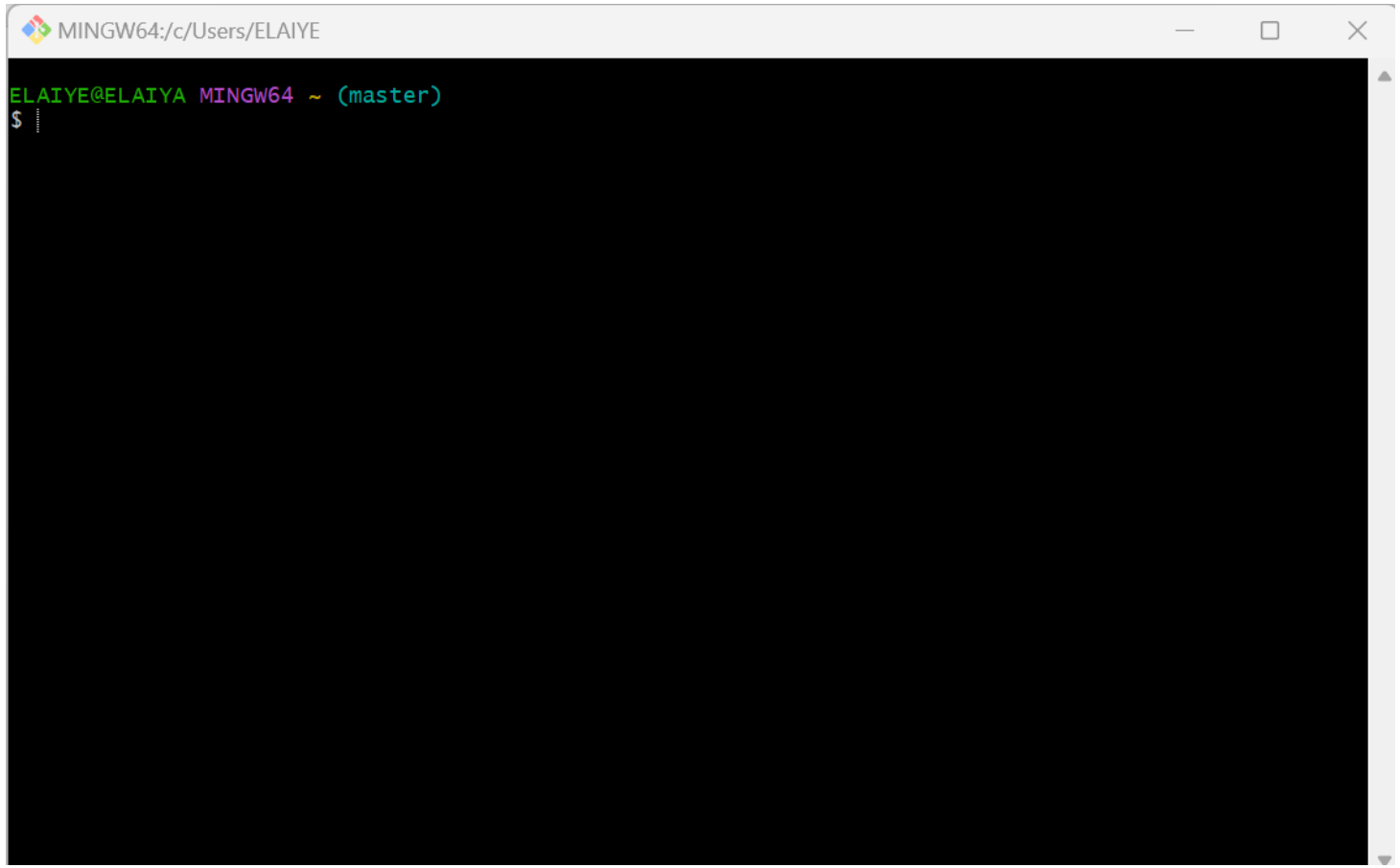


Click on Install



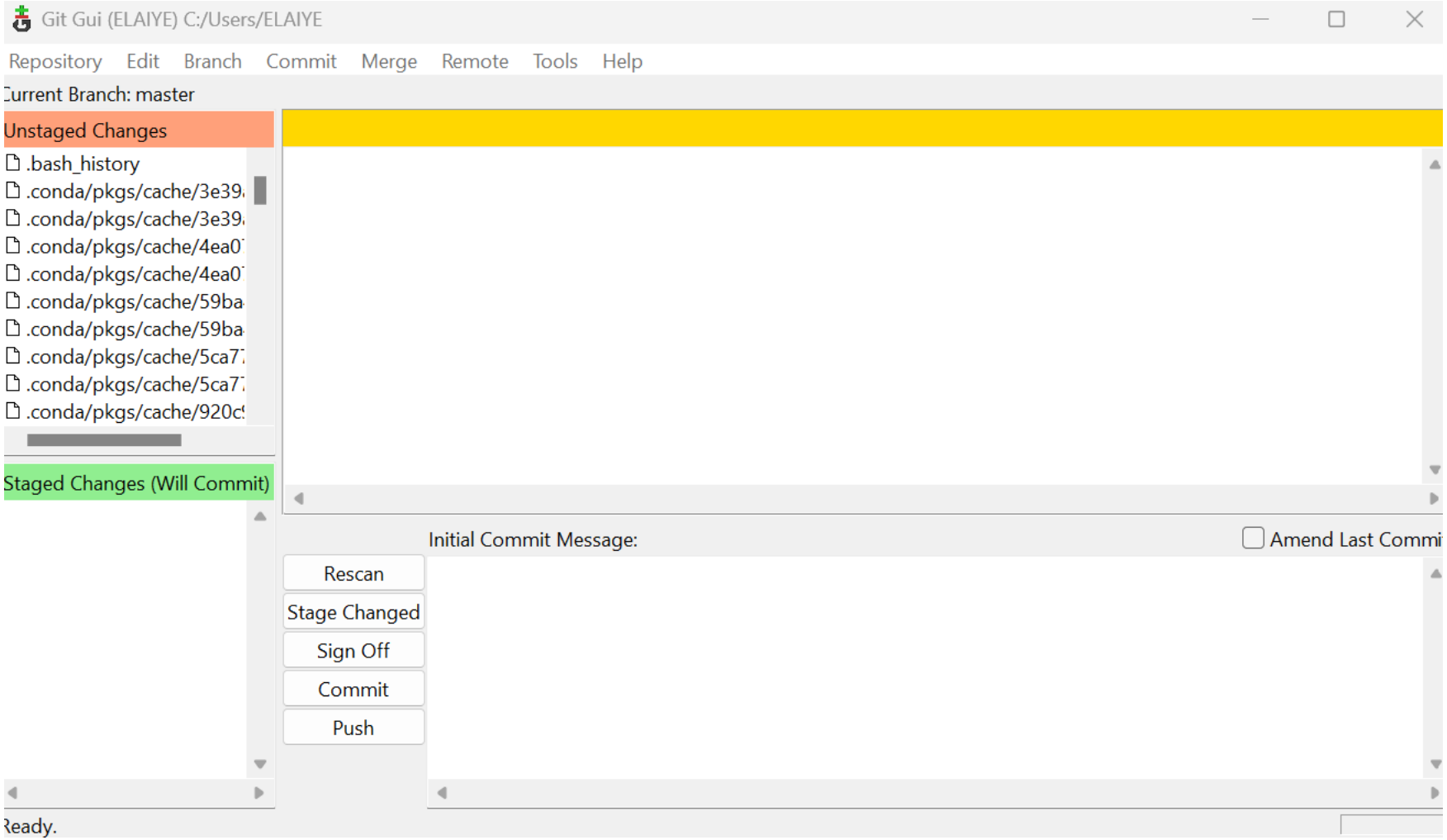
Click on Finish

BCS358C-Project Management with Git



Then you can view git bash Terminal

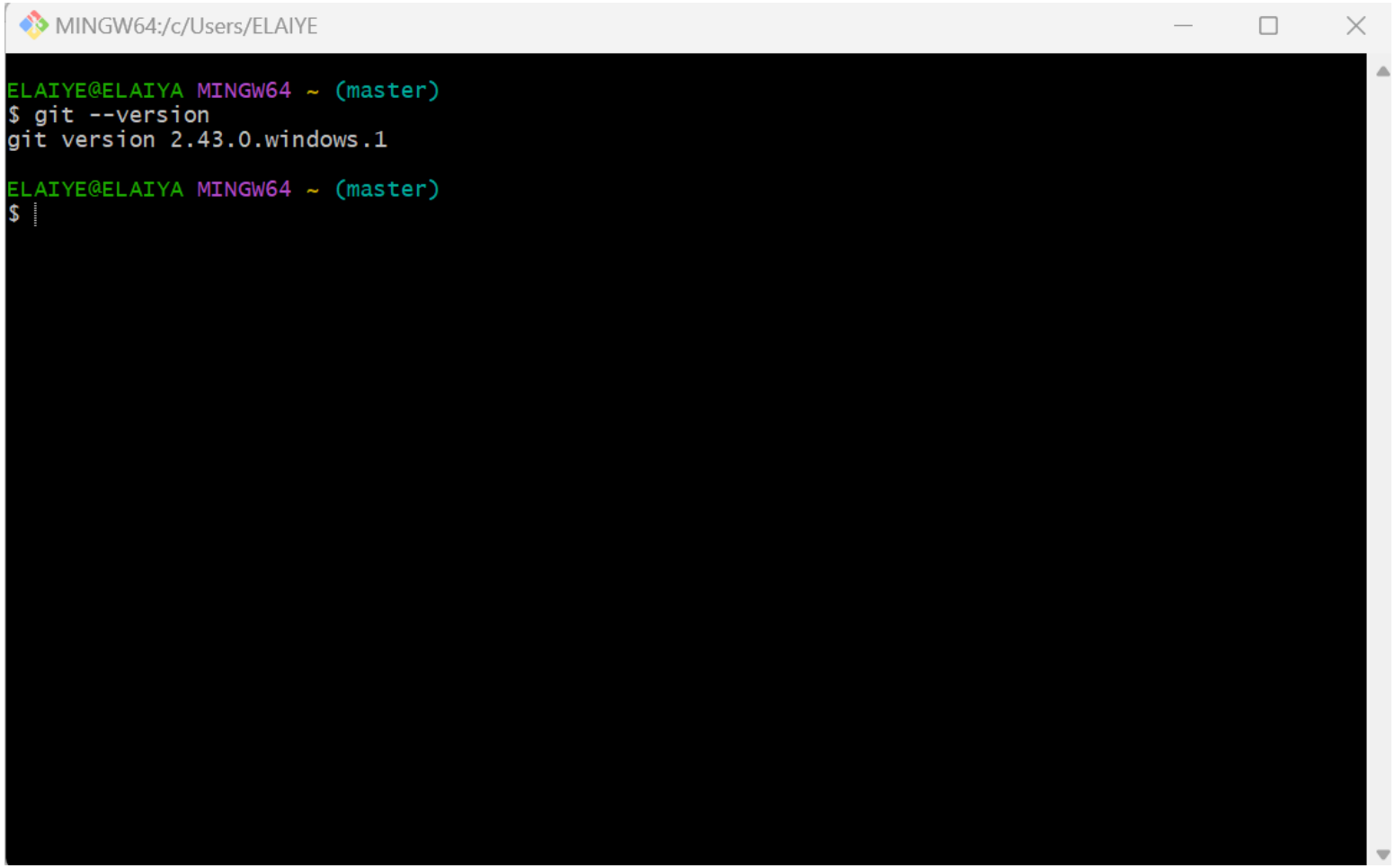
BCS358C-Project Management with Git



Then you can view git GUI

How to configure Git

BCS358C-Project Management with Git



Type

- git --version

You can view the version which is installed in the system

MINGW64:/c/Users/ELAIYE

```
ELAIYE@ELAIYA MINGW64 ~ (master)
$ git init
Reinitialized existing Git repository in C:/Users/ELAIYE/.git/

ELAIYE@ELAIYA MINGW64 ~ (master)
$
```

Type

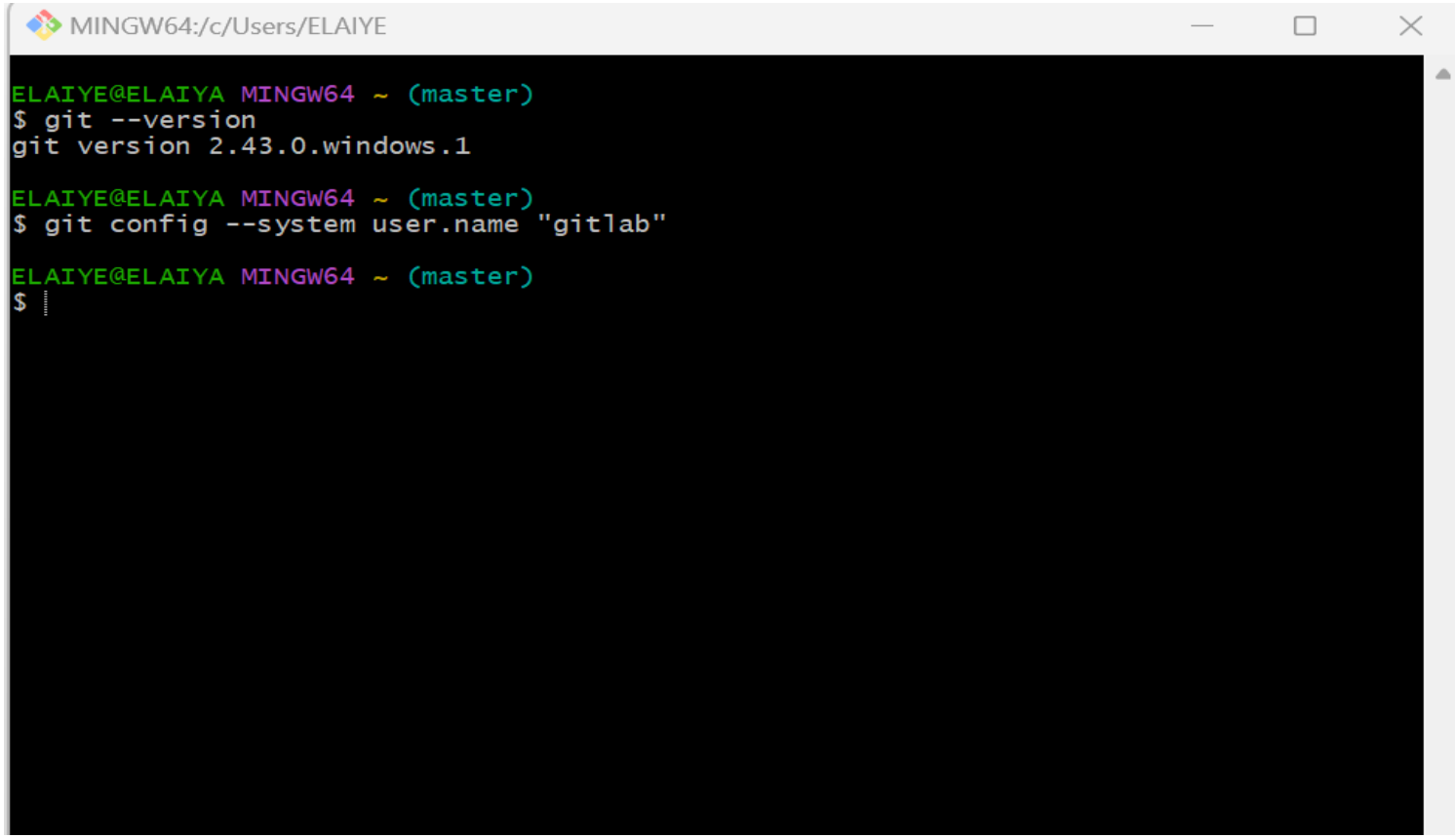
- git init

You can initialize git repository

BCS358C-Project Management with Git

To set your username, type and execute these commands:

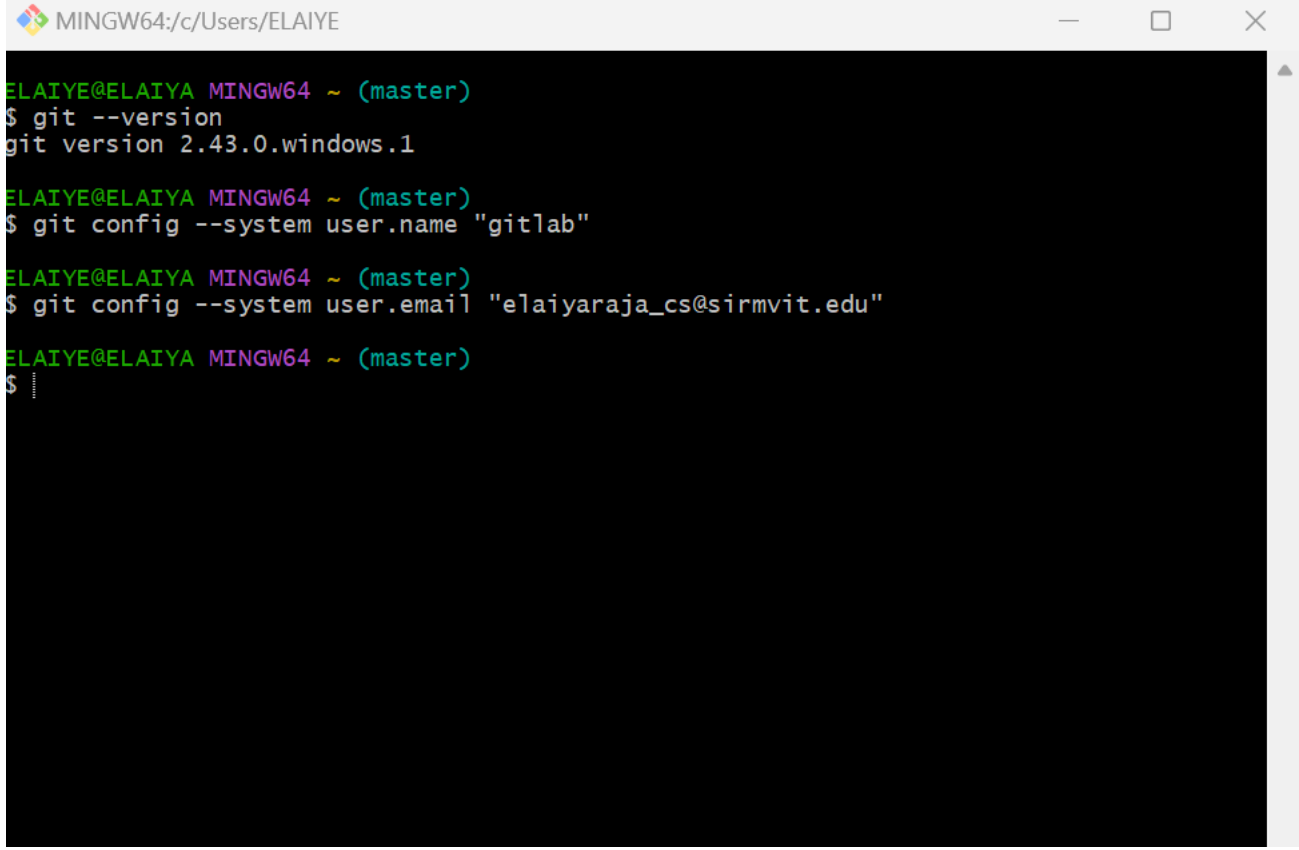
- git config --system user.name "gitlab"



BCS358C-Project Management with Git

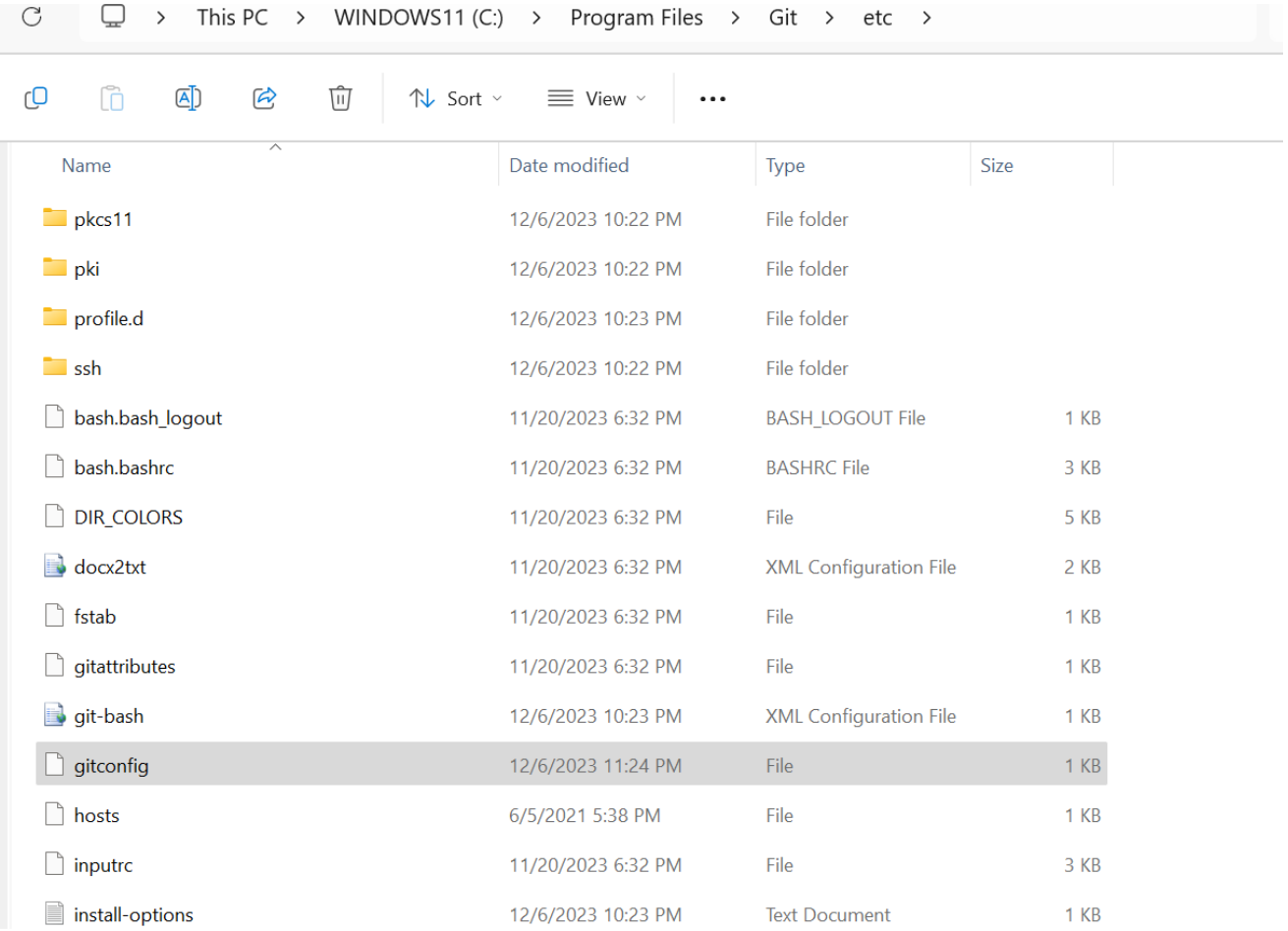
To set your username, type and execute these commands:

- git config --system user.email “elaiyaraja_cs@sirmvit.edu"



The user name get overwrites as your emailid as user name.

You can check in C:\Program Files\Git\etc



Open gitconfig Using notepad once and you can able to view in the master User name

BCS358C-Project Management with Git

MINGW64:/c/Users/ELAIYE/desktop

```
ELAIYE@ELAIYA MINGW64 ~ (master)
$ git init
Reinitialized existing Git repository in C:/Users/ELAIYE/.git/

ELAIYE@ELAIYA MINGW64 ~ (master)
$ cd desktop

ELAIYE@ELAIYA MINGW64 ~/desktop (master)
$
```

Get into the desktop

- Cd desktop

BCS358C-Project Management with Git

MINGW64:/c/Users/ELAIYE/desktop

```
ELAIYE@ELAIYA MINGW64 ~ (master)
$ git init
Reinitialized existing Git repository in C:/Users/ELAIYE/.git/

ELAIYE@ELAIYA MINGW64 ~ (master)
$ cd desktop

ELAIYE@ELAIYA MINGW64 ~/desktop (master)
$ mkdir gitlab

ELAIYE@ELAIYA MINGW64 ~/desktop (master)
$
```

Create directory

- mkdir gitlab

BCS358C-Project Management with Git

MINGW64:/c/Users/ELAIYE/desktop/gitlab

```
ELAIYE@ELAIYA MINGW64 ~ (master)
$ git init
Reinitialized existing Git repository in C:/Users/ELAIYE/.git/

ELAIYE@ELAIYA MINGW64 ~ (master)
$ cd desktop

ELAIYE@ELAIYA MINGW64 ~/desktop (master)
$ mkdir gitlab

ELAIYE@ELAIYA MINGW64 ~/desktop (master)
$ cd gitlab

ELAIYE@ELAIYA MINGW64 ~/desktop/gitlab (master)
$ ls

ELAIYE@ELAIYA MINGW64 ~/desktop/gitlab (master)
$
```

Get into directory

- cd gitlab

BCS358C-Project Management with Git

MINGW64:/c/Users/ELAIYE/desktop/gitlab

```
$ ls

ELAIYE@ELAIYA MINGW64 ~/desktop/gitlab (master)
$ ls
'gitlab class.txt'

ELAIYE@ELAIYA MINGW64 ~/desktop/gitlab (master)
$ git init
Initialized empty Git repository in C:/Users/ELAIYE/Desktop/gitlab/.git/

ELAIYE@ELAIYA MINGW64 ~/desktop/gitlab (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        gitlab class.txt

nothing added to commit but untracked files present (use "git add" to track)

ELAIYE@ELAIYA MINGW64 ~/desktop/gitlab (master)
$
```

Git can notify if your folder has some modified files

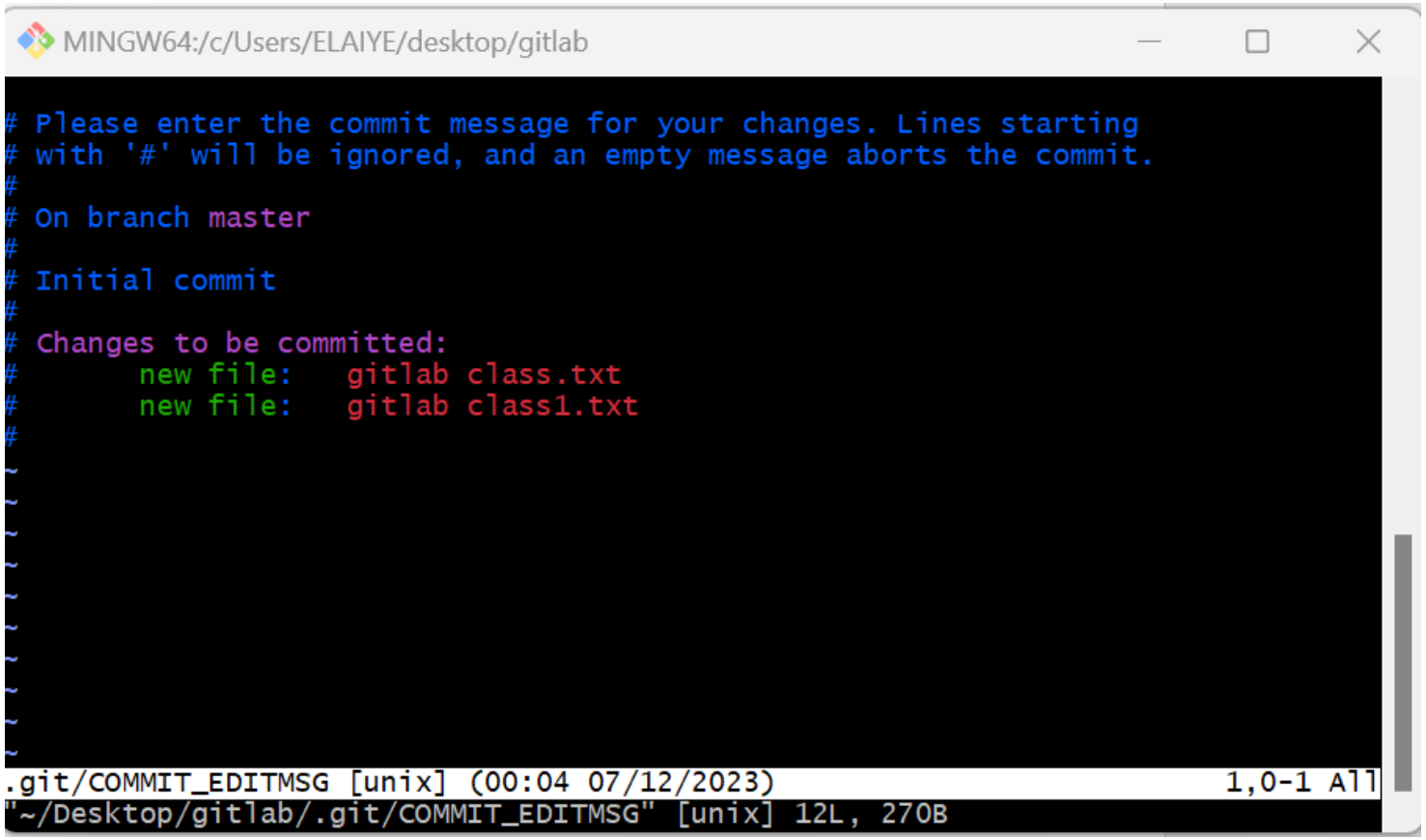
BCS358C-Project Management with Git

MINGW64:/c/Users/ELAIYE/desktop/gitlab

```
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        gitlab class.txt
        gitlab class1.txt

nothing added to commit but untracked files present (use "git add" to track)

ELAIYE@ELAIYA MINGW64 ~/desktop/gitlab (master)
$ git add gitlab class.txt
fatal: pathspec 'gitlab' did not match any files

ELAIYE@ELAIYA MINGW64 ~/desktop/gitlab (master)
$ git add "gitlab class.txt"

ELAIYE@ELAIYA MINGW64 ~/desktop/gitlab (master)
$ git add "gitlab class1.txt"

ELAIYE@ELAIYA MINGW64 ~/desktop/gitlab (master)
$
```

Select which files to commit

BCS358C-Project Management with Git



Type

- git commit

BCS358C-Project Management with Git

MINGW64:/c/Users/ELAIYE/desktop/gitlab

```
[master (root-commit) 601fd42] GIT LAB USERS
 2 files changed, 5 insertions(+)
 create mode 100644 gitlab class.txt
 create mode 100644 gitlab class1.txt

ELAIYE@ELAIYA MINGW64 ~/desktop/gitlab (master)
$
```

Type
Some content
Press Esc key
Type
:wq to save.

BCS358C-Project Management with Git

MINGW64:/c/Users/ELAIYE/desktop/gitlab

```
nothing to commit, working tree clean

ELAIYE@ELAIYA MINGW64 ~/desktop/gitlab (master)
$ commit
bash: commit: command not found

ELAIYE@ELAIYA MINGW64 ~/desktop/gitlab (master)
$ git commit
On branch master
nothing to commit, working tree clean

ELAIYE@ELAIYA MINGW64 ~/desktop/gitlab (master)
$ git diff

ELAIYE@ELAIYA MINGW64 ~/desktop/gitlab (master)
$ git log
commit 601fd4218982631d650530873c46d0b61e86a071 (HEAD -> master)
Author: gitlab <elaiyaraja_cs@sirmvit.edu>
Date:   Thu Dec 7 00:04:40 2023 +0530

    GIT LAB USERS

ELAIYE@ELAIYA MINGW64 ~/desktop/gitlab (master)
$
```

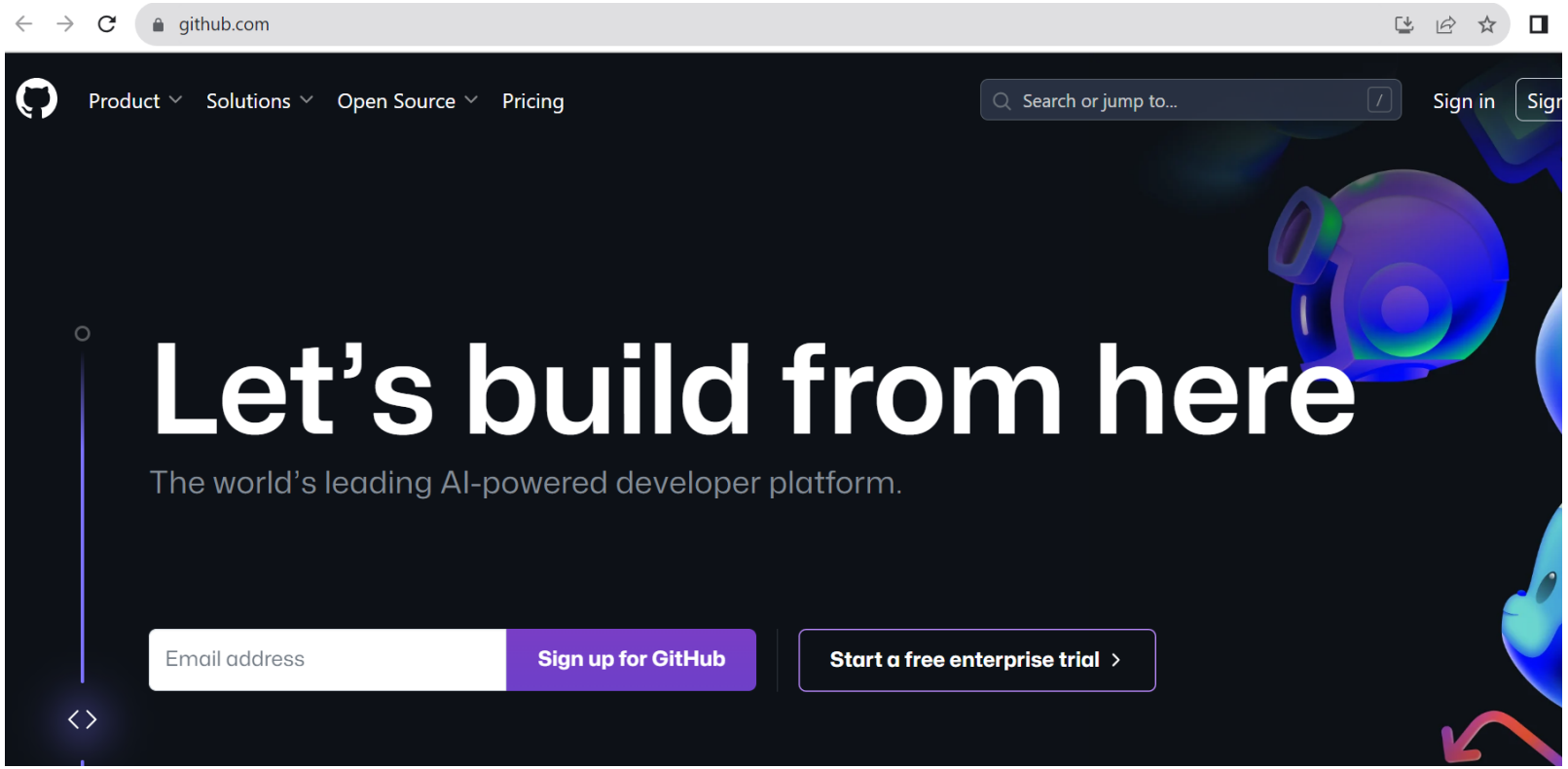
Type

- git log

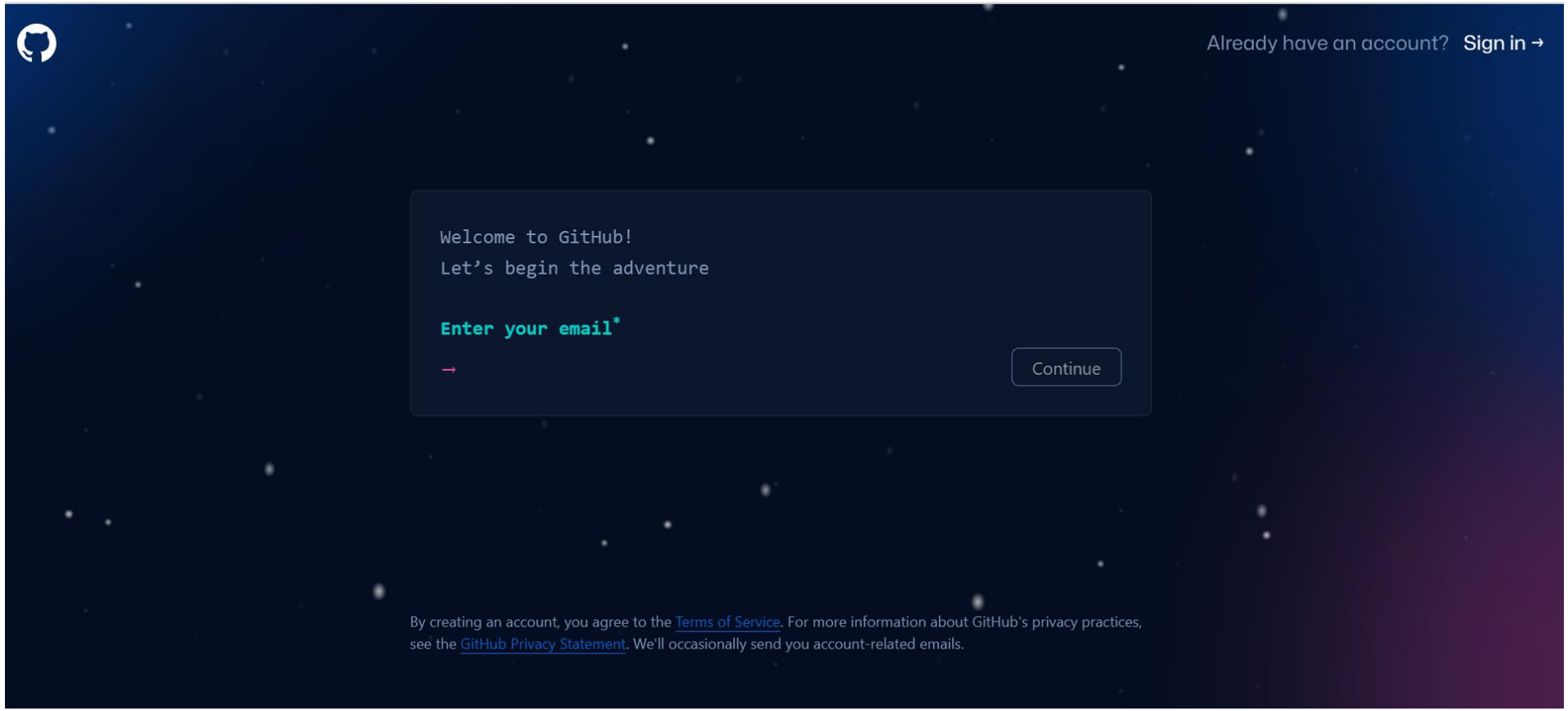
to check your logs

How to push a repository to GitHub

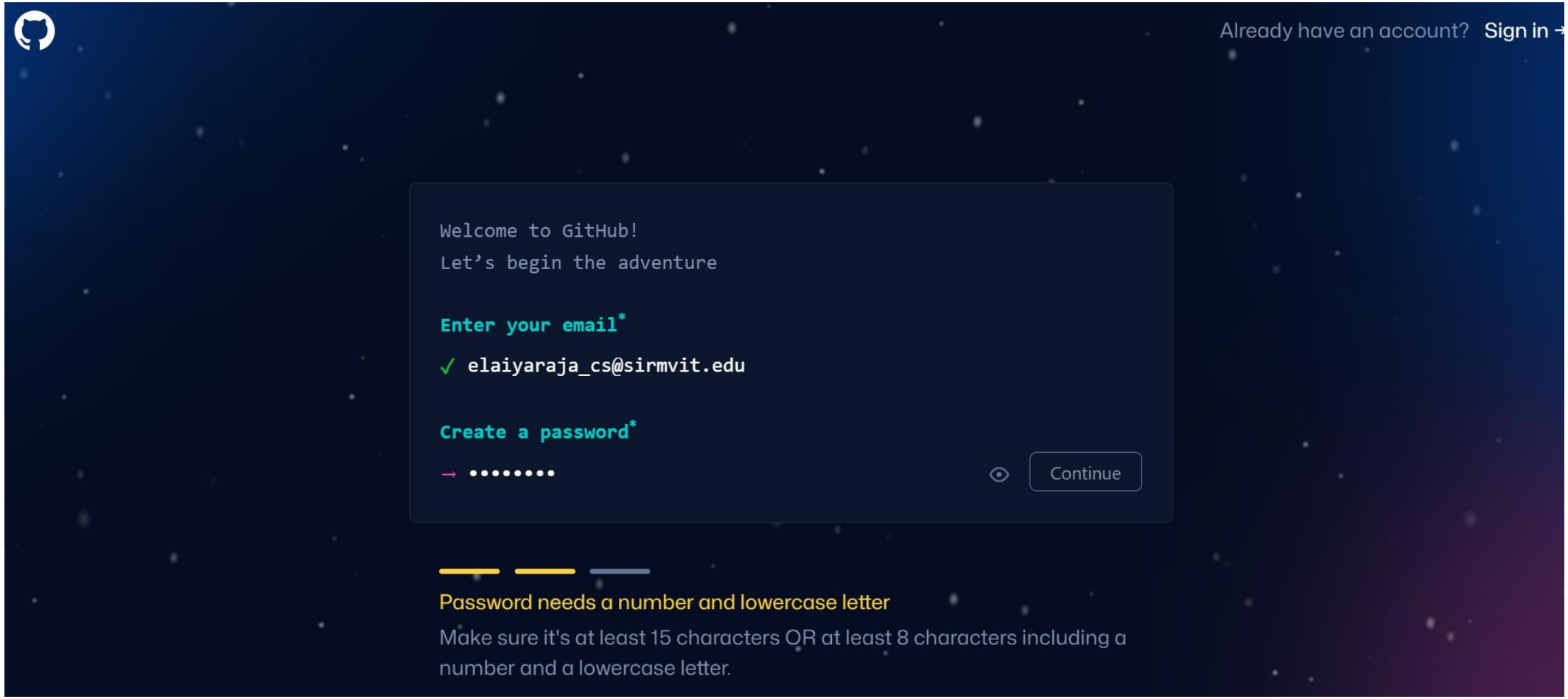
Step 1 – Create a GitHub account



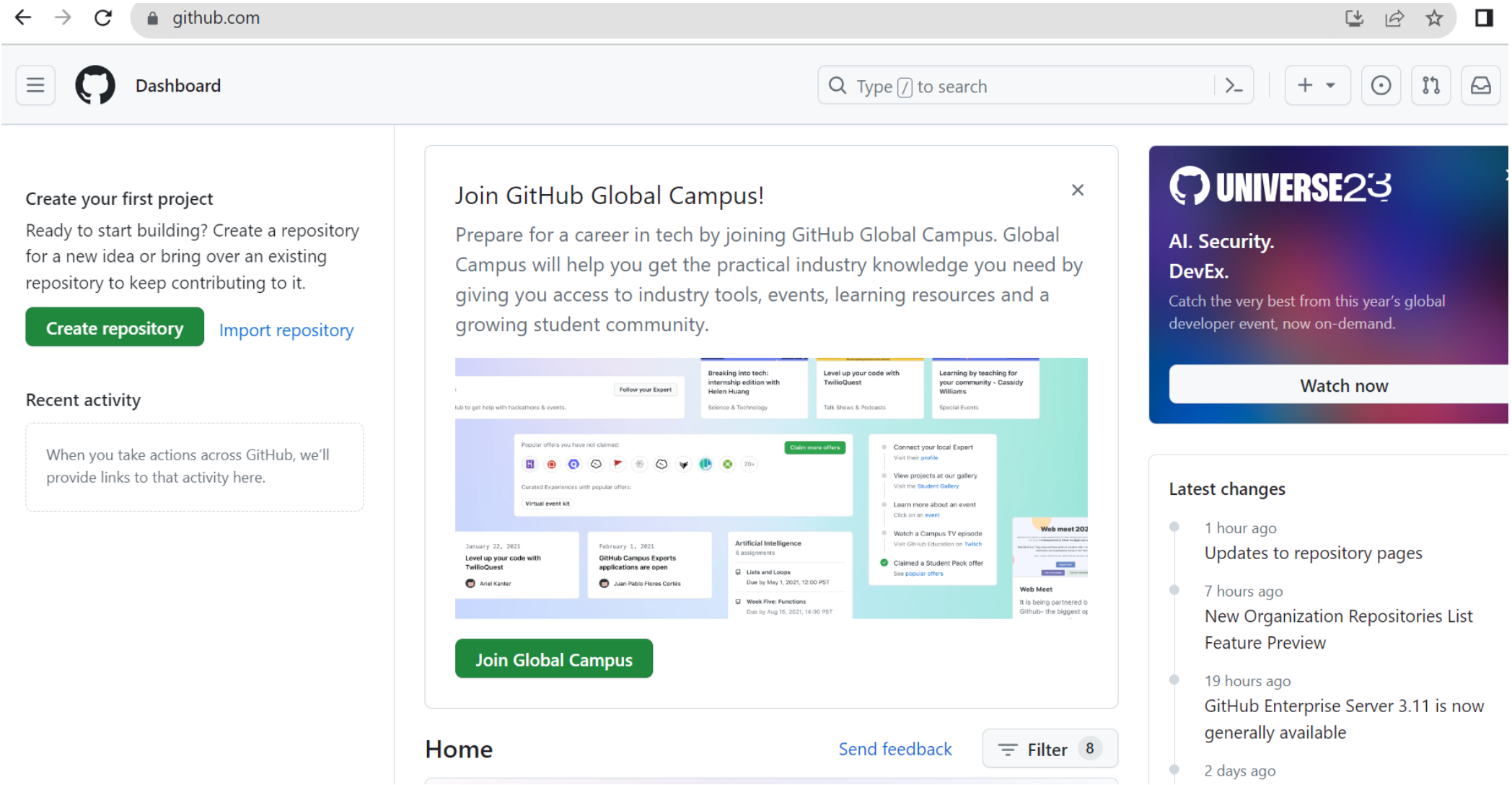
BCS358C-Project Management with Git



BCS358C-Project Management with Git

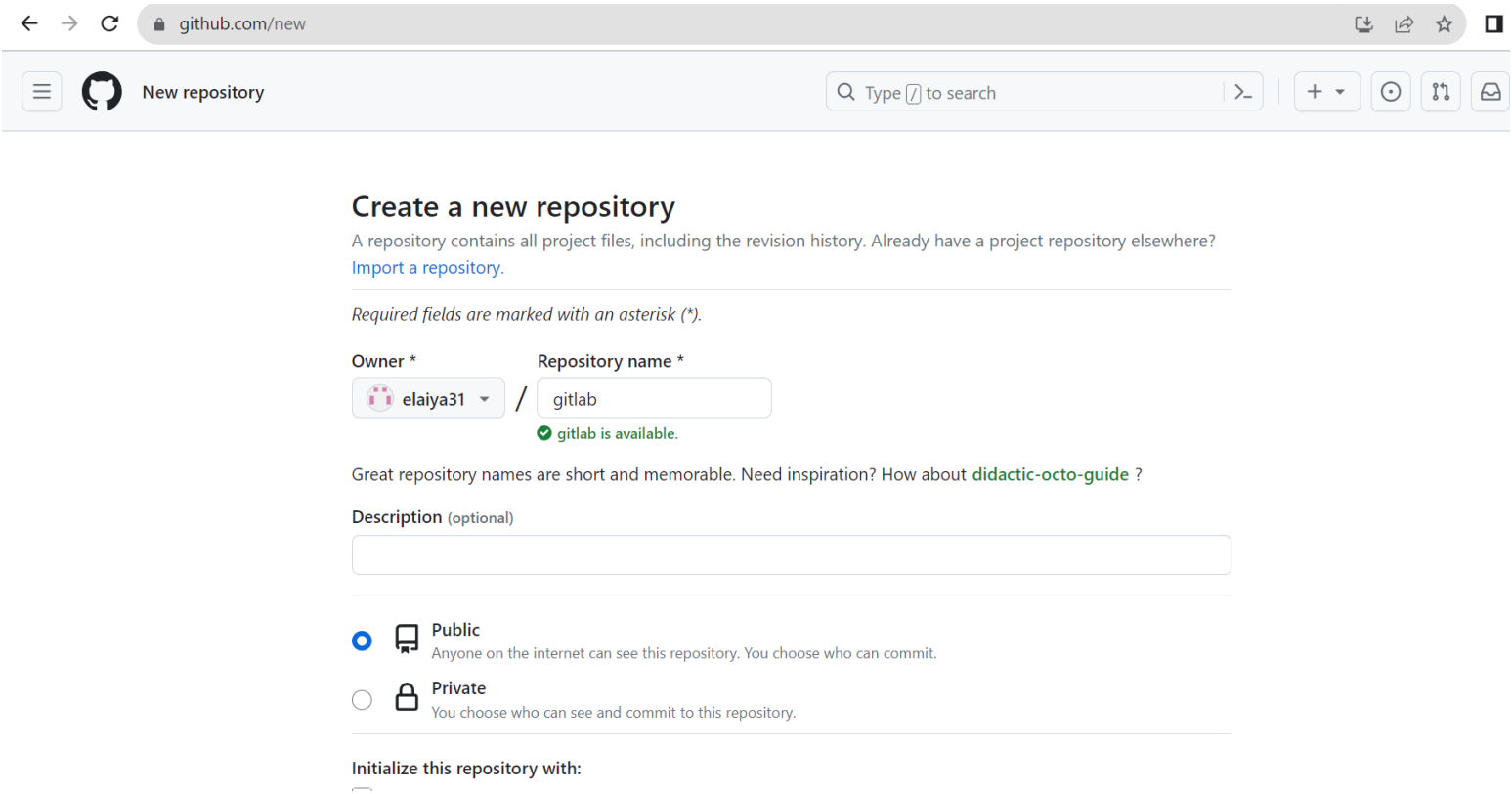


BCS358C-Project Management with Git



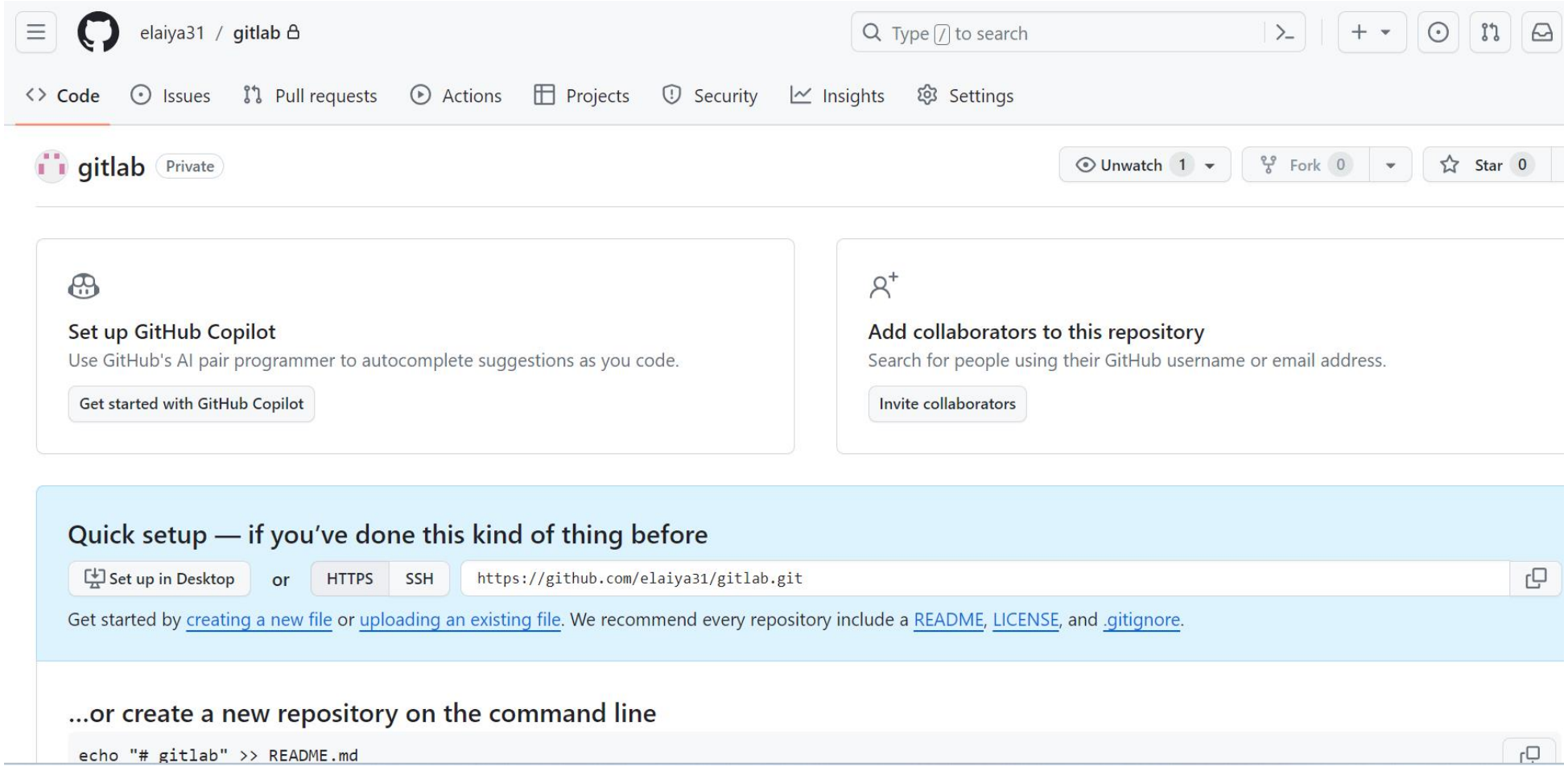
Step 2 – Create a repository

You can click on the + symbol on the top right corner of the page then choose "New repository". Give your repo a name then scroll down and click on "Create repository".



BCS358C-Project Management with Git

Step 2 – Create a repository



Basic commands

git init: Initializes a new Git repository in the current directory.

git clone <repository URL>: Creates a copy of a remote repository on your local machine.

git add <file>: Stages a file to be committed, marking it for tracking in the next commit.

git commit -m "message": Records the changes you've staged with a descriptive commit message.

git status: Shows the status of your working directory and the files that have been modified or staged.

git log: Displays a log of all previous commits, including commit hashes, authors, dates, and commit messages.

git diff: Shows the differences between the working directory and the last committed version.

git branch: Lists all branches in the repository and highlights the currently checked- out branch.

git branch <branchname>: Creates a new branch with the specified name.

git checkout <branchname>: Switches to a different branch.

git merge <branchname>: Merges changes from the specified branch into the currently checked-out branch.

EXPERIMENT 1 :

Setting Up and Basic Commands

Initialize a new Git repository in a directory. Create a new file and add it to the staging area and commit the changes with an appropriate commit message.

$ git init

1. Create a new file in the directory. For example, let's create a file named "my_file.txt." You can use any text editor or command-line tools to create the file.
2. Add the newly created file to the staging area. Replace "my_file.txt" with the actual name of your file:

$ git add my_file.txt

This command stages the file for the upcoming commit.

$ git commit -m "Your commit message here"

Your commit message should briefly describe the purpose or nature of the changes you made. For example:

$ git commit -m "Add a new file called my_file.txt"

After these steps, your changes will be committed to the Git repository with the provided commit message. You now have a version of the repository with the new file and its history stored in Git.

EXPERIMENT 2 :

Creating and Managing Branches

Create a new branch named "feature-branch." Switch to the "master" branch. Merge the "feature-branch" into "master."

Creating and Managing Branches:

Create a new branch named "feature branch.”Switch to the" master" branch. Merge the "feature-branch" into "master."

Solution:

To create a new branch named "feature-branch," switch to the "master" branch, and merge the "feature-branch" into "master" in Git, follow these steps:

1. Make sure you are in the "master" branch by switching to it:

$ git checkout master

1. Create a new branch named "feature-branch" and switch to it:

$ git checkout -b feature-branch

This command will create a new branch called "feature-branch" and switch to it.

1. Make your changes in the "feature-branch" by adding, modifying, or deleting files as needed.
2. Stage and commit your changes in the "feature-branch":

git add .

$ git commit -m "Your commit message for feature-branch"

Replace "Your commit message for feature-branch" with a descriptive commit message for the changes you made in the "feature-branch."

1. Switch back to the "master" branch:

$ git checkout master

1. Merge the "feature-branch" into the "master" branch:

$ git merge feature-branch

This command will incorporate the changes from the "feature-branch" into the "master" branch.

Now, your changes from the "feature-branch" have been merged into the "master" branch. Your project's history will reflect the changes made in both branches

EXPERIMENT 3 :

Creating and Managing Branches

Write the commands to stash your changes, switch branches, and then apply the stashed changes.

Creating and Managing Branches:

Write the commands to stash your changes, switch branches, and then apply the stashed changes.

Solution:

To stash your changes, switch branches, and then apply the stashed changes in Git, you can use the following commands:

1. Stash your changes:

$ git stash save "Your stash message"

This command will save your changes in a stash, which acts like a temporary storage for changes that are not ready to be committed.

1. Switch to the desired branch:

$ git checkout target-branch

Replace "target-branch" with the name of the branch you want to switch to.

1. Apply the stashed changes:

$ git stash apply

This command will apply the most recent stash to your current working branch. If you have multiple stashes, you can specify a stash by name or reference (e.g., git stash apply stash@{2}) if needed.

If you want to remove the stash after applying it, you can use git stash pop instead of git stash apply.

Remember to replace "Your stash message" and "target-branch" with the actual message you want for your stash and the name of the branch you want to switch to.

EXPERIMENT 4 :

Collaboration and Remote Repositories:

Clone a remote Git repository to your local machine.

BCS358C-Project Management with Git

1. Open your terminal or command prompt.
2. Navigate to the directory where you want to clone the remote Git repository. You can use the cd command to change your working directory.
3. Use the git clone command to clone the remote repository. Replace <repository_url> with the URL of the remote Git repository you want to clone. For example, if you were cloning a repository from GitHub, the URL might look like this:

$ **git clone <repository_url>**

$ git clone https://github.com/username/repo-name.git
Ex: https://github.com/elaiya31/3githubnotes

1. Git will clone the repository to your local machine. Once the process is complete, you will have a local copy of the remote repository in your chosen directory.

You can now work with the cloned repository on your local machine, make changes, and push those changes back to the remote repository as needed.

EXPERIMENT 5 :

Collaboration and Remote Repositories:

Fetch the latest changes from a remote repository and rebase your local branch onto the updated remote branch.

1. Open your terminal or command prompt.
2. Make sure you are in the local branch that you want to rebase. You can switch to the branch using the following command, replacing <branch-name> with your actual branch name:

$ git checkout <branch-name>

1. Fetch the latest changes from the remote repository. This will update your local repository with the changes from the remote without merging them into your local branch:

$ git fetch origin

Here, origin is the default name for the remote repository. If you have multiple remotes, replace origin with the name of the specific remote you want to fetch from.

1. Once you have fetched the latest changes, rebase your local branch onto the updated remote branch:

$ **git rebase origin/<branch-name>**

Replace <branch-name> with the name of the remote branch you want to rebase onto. This command will reapply your local commits on top of the latest changes from the remote branch, effectively incorporating the remote changes into your branch history.

1. Resolve any conflicts that may arise during the rebase process. Git will stop and notify you if there are conflicts that need to be resolved. Use a text editor to edit the conflicting files, save the changes, and then continue the rebase with:

$ git rebase --continue

1. After resolving any conflicts and completing the rebase, you have successfully updated your local branch with the latest changes from the remote branch.
2. If you want to push your rebased changes to the remote repository, use the git push command. However, be cautious when pushing to a shared remote branch, as it can potentially overwrite other developers' changes:

$ git push origin <branch-name>

Replace <branch-name> with the name of your local branch.By following these steps, you can keep your local branch up to date with the latest changes from the remote repository and maintain a clean and linear history through rebasing.

EXPERIMENT 6

Collaboration and Remote Repositories:

Write the command to merge "feature-branch" into "master" while providing a custom commit message for the merge.

BCS358C-Project Management with Git

To merge the "feature-branch" into "master" in Git while providing a custom commit message for the merge, you can use the following command:

$ **git checkout master**
$ git merge feature-branch -m "Your custom commit message here"

Replace "Your custom commit message here" with a meaningful and descriptive commit message for the merge. This message will be associated with the merge commit that is created when you merge "feature-branch" into "master."

EXPERIMENT 7.

Git Tags and Releases:

Write the command to create a lightweight Git tag named "v1.0” for a commit in your local repository.

To create a lightweight Git tag named "v1.0" for a commit in your local repository, you can use the following command:

$ git tag v1.0

This command will create a lightweight tag called "v1.0" for the most recent commit in your current branch. If you want to tag a specific commit other than the most recent one, you can specify the commit's SHA-1 hash after the tag name. For example:

$ **git tag v1.0 <commit-SHA>**

Replace <commit-SHA> with the actual SHA-1 hash of the commit you want to tag.

Experiment 8.

Advanced Git Operations:

Write the command to cherry-pick a range of commits from "source-branch" to the current branch.

To cherry-pick a range of commits from "source-branch" to the current branch, you can use the following command:

$git cherry-pick <start-commit>^..<end-commit>

Replace <start-commit> with the commit at the beginning of the range, and <end-commit> with the commit at the end of the range. The ^ symbol is used to exclude the <start-commit> itself and include all commits after it up to and including <end-commit>. This will apply the changes from the specified range of commits to your current branch.

For example, if you want to cherry-pick a range of commits from "source-branch" starting from commit ABC123 and ending at commit DEF456, you would use:

$ git cherry-pick ABC123^..DEF456

Make sure you are on the branch where you want to apply these changes before running the cherry-pick command.

Experiment 9.

Analysing and Changing Git History:

Given a commit ID, how would you use Git to view the details of that specific commit, including the author, date, and commit message?

To view the details of a specific commit, including the author, date, and commit message, you can use the git show or git log command with the commit ID. Here are both options:

1. Using git show:

```
bash
git show <commit-ID>
```

Replace <commit-ID> with the actual commit ID you want to view. This command will display detailed information about the specified commit, including the commit message, author, date, and the changes introduced by that commit.

For example:

$ git show abc123

1. Using git log:

$ git log -n 1 <commit-ID>

The -n 1 option tells Git to show only one commit. Replace <commit-ID> with the actual commit ID. This command will display a condensed view of the specified commit, including its commit message, author, date, and commit ID.

For example:

$ git log -n 1 abc123

Both of these commands will provide you with the necessary information about the specific commit you're interested in.

Experiment 10.

Analysing and Changing Git History:

Write the command to list all commits made by the author "JohnDoe" between "2023- 01-01"and "2023-12-31."

To list all commits made by the author "JohnDoe" between "2023-01-01" and "2023-12-31" in Git, you can use the git log command with the --author and --since and --until options. Here's the command:

$ git log --author="JohnDoe" --since="2023-01-01" --until="2023-12-31"

This command will display a list of commits made by the author "JohnDoe" that fall within the specified date range, from January 1, 2023, to December 31, 2023. Make sure to adjust the author name and date range as needed for your specific use case.

Experiment 11.

Analysing and Changing Git History:

Write the command to display the last five commits in the repository's history.

To display the last five commits in a Git repository's history, you can use the git log command with the -n option, which limits the number of displayed commits. Here's the command:

$ git log -n 5

This command will show the last five commits in the repository's history. You can adjust the number after -n to display a different number of commits if needed.

Experiment 12.

Analysing and Changing Git History:

Write the command to undo the changes introduced by the commit with the ID "abc123".

To undo the changes introduced by a specific commit with the ID "abc123" in Git, you can use the git revert command. The git revert command creates a new commit that undoes the changes made by the specified commit, effectively "reverting" the commit.

Here's the command:

$ git revert abc123

Replace "abc123" with the actual commit ID that you want to revert. After running this command, Git will create a new commit that negates the changes introduced by the specified commit. This is a safe way to undo changes in Git because it preserves the commit history and creates a new commit to record the reversal of the changes.

BCS358C-Project Management with Git

TEST : LAB INTERNAL - I

Subject Code : BCS358C

Subject Name : Project Management with Git

1	**3RD SEMESTER-A SEC.**	**B1**	**05/03/2024**	**9.30 TO 11.30 AM**
2		**B2**	**05/03/2024**	**1.35 TO 3.35 PM**

THE END